REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	11.Jan.99	DISSERTATION

4. TITLE AND SUBTITLE
PERFORMANCE ANALYSIS OF THE SIMULTANEOUS OPTICAL MULTIPORCESSOR EXCHANGE BUS ARCHITECTURE

5. FUNDING NUMBERS

6. AUTHOR(S)
MAJ DOSKOCZ EDWARD K

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSITY OF ALABAMA HUNTSVILLE

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
FY99-47

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS	15. NUMBER OF PAGES	16. PRICE CODE
	118	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

PERFORMANCE ANALYSIS OF THE SIMULTANEOUS OPTICAL MULTIPROCESSOR EXCHANGE BUS ARCHITECTURE

by

EDWARD KOPLIN DOSKOCZ

A DISSERTATION

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy in

The Department of Electrical and Computer Engineering

of

The School of Graduate Studies

of

The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

1998

DISSERTATION APPROVAL FORM

Submitted by Edward K. Doskocz in partial fulfillment of the requirements for the degree of Doctor of Philosophy with a major in Computer Engineering.

Accepted on behalf of the Faculty of the School of Graduate Studies by the dissertation committee:

8/14/98 Committee Chair
(Date)

8/14/98 Department Chair

R. M. Mychalkiw 8/17/98 College Dean

J. D. Stanner 8/17/98 Graduate Dean

ABSTRACT
School of Graduate Studies
University of Alabama in Huntsville

Degree Doctor of Philosophy College/Dept. Engineering / Computer

Name of Candidate Edward Koplin Doskocz

Title Performance Analysis of the Simultaneous Optical Multiprocessor Exchange Bus Architecture

The computing world has entered an era in which executing high-performance applications has become commonplace in the scientific, engineering, and business workplace. The trend in these applications is toward the execution of larger or more detailed models in less time, with parallel processing seen as a useful tool for performing this work. The focus of this dissertation is the performance evaluation of a parallel-processing architecture, known as the Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus), that exploits the parallelism in these applications via efficient data communication between processors. The SOME-Bus is a low-latency, high-bandwidth, broadcast-based fiber-optic interconnection network which can efficiently interconnect over one hundred processing nodes.

Performance potential of the SOME-Bus architecture is examined by comparing its quantifiable network characteristics to those of existing static and dynamic network topologies. The need for a broadcast-based parallel-processing design is examined by reviewing research into the workload, communication patterns, and performance of current parallel systems. Theoretical and simulation models are developed for both message-passing and distributed-shared-memory parallel processing paradigms. The message-passing

model consists of a closed queueing network. An efficient solution method is developed using Norton's Theorem for Queueing Networks. Two distributed-shared-memory models are constructed, one based on a closed, multi-class queueing network, the other on a Markov-chain. Mean Value Analysis is used to evaluate the first model; the second model is evaluated by examining state probabilities. Performance results from the models and simulations are compared to results from crossbar and torus system simulations. Models and simulators were implemented using MATLAB and C languages.

Results indicate that quantifiable SOME-Bus interconnection-network characteristics equal or exceed the best available from existing static and dynamic network topologies. A review of current research shows that the existence of a broadcast-based interconnection network, such as the SOME-Bus, is justified due to the significant number of broadcast operations that commonly occur in high-performance parallel applications. Results from theoretical and simulation studies show that SOME-Bus performance, at a minimum, equals the best available from torus and crossbar systems operating in either a message-passing or distributed-shared-memory paradigm. In the most realistic operational scenarios, the SOME-Bus appears to have a significant performance advantage over these two popular architectures.

Abstract Approval:

Committee Chair C. Li 8/14/98
(Date)

Department Chair Ben Ochia 8/14/98

Graduate Dean J. D. Flamer 8/17/98

ACKNOWLEDGEMENTS

The work described in this dissertation would not have been possible without the assistance of the Air Force, and a number of people who deserve special mention. It was sponsorship by the Air Force Academy under a program administered by the Air Force Institute of Technology, that allowed me to pursue this research. I would like to thank Dr. Constantine Katsinis for offering this topic, and for the patient guidance he provided throughout the work. The research of Dr. Kulick and Dr. Gaede, among others, has given me the insight into the SOME-Bus optical network and receiver designs that was necessary to complete my work. In addition, the members of my committee have been very helpful from both a technical, and an administrative, standpoint.

Finally, I want to offer a special thanks to my wife, Mali, for giving me her full support throughout this endeavor; this accomplishment would not have been possible without it.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

LIST OF SYMBOLS

Symbol	Definition
B	Memory bandwidth in bytes/sec.
c	Channel designator.
C	Number of service centers in a chain (DSM).
$C(K)$	Normalization constant.
$C_r(K)$	Normalization constant for reference node.
$C_{Net}(k)$	Network normalization constants.
D	Number of data messages preceding a synchronization operation.
$\mathbf{e}_r$	Unit vector with the 1 in the r^{th} position.
h	Mean time interval between remote memory requests (DSM).
$h_i(k_i)$	Unnormalized queue-length distribution of service center i.
K	Total number of messages(processes) in a message-passing system.
K	Number of threads processed at a DSM system node.
$\mathbf{K}$	Chain population vector (DSM).
$\mathbf{K}_{i_j}$	Vector of chain j message population at service center i (DSM).
k_i	Number of messages at service center i (message-passing).
k_i	Outstanding remote-memory requests by node i in a DSM system.
$\bar{k}_i(K)$	Average number of messages at center i, given K messages.
m	Mean time interval for DMA processing (DSM).
M	Number of bytes in a message.
n	Number of nodes in one dimension of a torus or mesh system.

Symbol	Definition
N	Number of nodes in a system.
n_{i_j}	Mean number of chain j messages at center i (DSM).
$n_{i_j}(\mathbf{r}-)$	Mean number of chain j messages in service center i just prior to the arrival of a message from chain r.
p	Processor designator.
P	Number of processors in a system.
$P(k_1,k_2,...k_N)$	State probability given the population vector $(k_1,k_2,...k_N)$.
P_{TR}	Set of states with messages in both processor queues (DSM).
P_{TO}	Set of states with only response queue occupied (DSM).
P_{OR}	Set of states with only request queue occupied (DSM).
$\mathbf{Q}$	The transition-rate matrix (DSM).
r	Reference-node designator.
s	Mean time interval for message transmission (DSM).
S	Set of states in the Markov-chain state space (DSM).
t_c	Average channel service time in a message-passing system.
t_{i_r}	Mean time chain r message spends at service center i (DSM).
t_p	Average processor service time in a message-passing system.
$\bar{t}_i(K)$	Average time spent at center i, given K messages in the network.
U_i	Utilization of service center i (message-passing).
$util_T$	Thread-processing processor utilization (DSM).
$util_R$	DMA-processing processor utilization (DSM).
w	Bus width (bits).

Symbol	Definition
$\mathbf{X}$	Transition probability matrix.
x_{ab}	Transition probability from node *a* to node *b*.
x_{ba}	Transition probability from node *b* to node *a*.
x_{ij}	Probability a message will transition from center i to center j.
$x_{i\sigma_2}$	Probability of transitioning from node i to the aggregate center.
$x_{\sigma_2 j}$	Probability of transitioning from node j to the aggregate center.
$Y_r(K)$	Reference node throughput.
α_T	Fraction of available service for thread processing (DSM).
α_R	Fraction of available service for DMA processing (DSM).
γ_a	Probability of a message traversing the branch into node a.
γ_b	Probability of a message traversing the branch into node b.
$\varepsilon_{i_j}(\mathbf{r}-)$	Difference in the average number of chain *j* messages at station *i*.
$\boldsymbol{\lambda}$	Equilibrium arrival (departure) rate vector.
λ_{c_r}	Channel center arrival rate at reference node.
λ_i	Message-passing departure rate from service center i.
λ_j	Message-passing arrival rate at service center j.
λ_j	Throughput of chain *j* (DSM).
λ_{p_r}	Processor center arrival rate at reference node.
μ_{c_r}	Service rate of reference node channel center.
μ_i	Load-independent service rate at service center i.
$\mu_i(k)$	Load-dependent service rate at service center i.

Symbol	Definition
μ_{p_r}	Service rate of reference node processor center.
π	The state-probability vector (DSM).
σ_1	Subsystem of interest (Norton's Theorem for Queueing Networks).
σ_2	Remainder of network (Norton's Theorem for Queueing Networks).
τ_{i_j}	Service time at center i for a chain j message (DSM).
$\hat{\tau}_i$	Effective service rate for chain *r* message at service center *i* (DSM).
τ_{i_j}	Average service time at center i for chain j messages (DSM).

Chapter 1

INTRODUCTION

1.1 Background

The computing world has entered an era in which executing high-performance applications, defined as computation-intensive, data-intensive, or both, has become commonplace in the scientific, engineering, and business workplace [50], [10], [57]. The trend in these applications is toward the execution of larger or more detailed models in less time, with parallel processing seen as a useful tool for performing this work. Exploiting the parallelism in these applications requires efficient data communication between processors. The focus of this dissertation is the performance evaluation of a parallel-processing architecture that accomplishes this task, the Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus). It is a low-latency, high-bandwidth, broadcast-based fiber-optic interconnection network which can efficiently interconnect over one hundred processor nodes, directly linking arbitrary pairs without contention [26], [28], [35].

Before a full-scale performance evaluation of the SOME-Bus is undertaken, it is important to determine if the study is warranted. A summary of SOME-Bus architectural features is followed by a comparison of its quantifiable network characteristics to those of existing systems to prove its performance potential. With performance potential established, the need for a broadcast-

based SOME-Bus design will be demonstrated. The framework for both of these undertakings exists in Gordon Bell's taxonomy of the multiple-instruction, multiple-data (MIMD) category of computer architecture established by Michael Flynn [4], [22]. Bell divides MIMD systems into multicomputers and multiprocessors, and then cites examples of each based on network topology. The static and dynamic characteristics of those networks provides the comparative basis for determining SOME-Bus performance potential. Concluding this chapter, a review of research into the workload, communication patterns, and performance of existing multicomputer and multiprocessor systems will demonstrate the need for a broadcast-based network.

1.2 SOME-Bus Architectural Summary

The Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus) is a parallel-processor interconnection network capable of interconnecting over 100 processing nodes [26] ,[28], [35]. It features high bandwidth (scaling directly with the number of nodes), low latency, no arbitration delay, and non-blocking, broadcast-based communication. SOME-Bus processing nodes communicate with each other through a wavelength-division-multiplexed, optical-interconnection network. The network is implemented with laser-diode transmitters, optical fiber transmission lines with induced Bragg gratings, and receivers formed of CMOS devices with integrated amorphous silicon detector superstructures. The Bragg gratings in the optical fiber serve as narrow-band, inexpensive optocouplers between the network and the receivers.

The SOME-Bus architecture for an *N*-node system is illustrated in Figure 1.1. Each node contains a receiver array (with buffering), a processing element (processor/memory), and a transmitter. The dedicated data transmission channel for each processor eliminates the need for network arbitration, and provides a system where bandwidth scales directly with the number of processors. The receiver array at each node contains *N* receivers, one dedicated to each data transmission channel. This design eliminates blocking since there is no contention for shared resources.

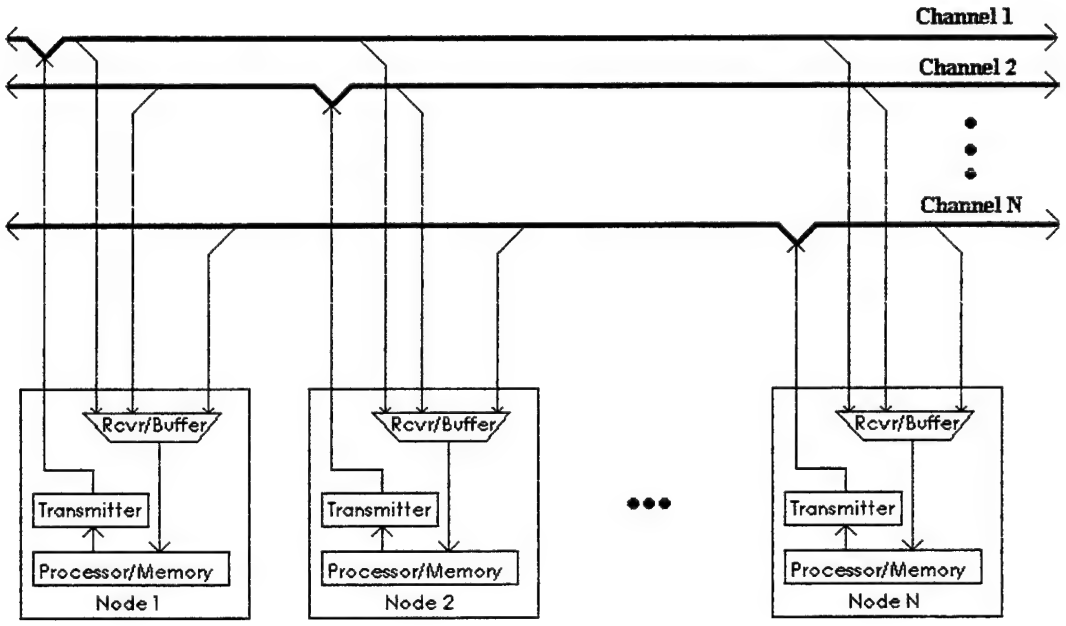


Figure 1.1 SOME-Bus Architecture

The transmitter subsystem is implemented with laser diodes and signal insertion hardware [35]. This hardware allows up to four processors, each using a dedicated wavelength, to insert data streams into a single fiber by employing

wavelength-division-multiplexing. In addition, a separate wavelength/fiber is used for a clock signal which is common to all data channels on that fiber. There is one insertion-point per fiber, dividing the fiber into two parts: one part services processors to the left of the insertion point, the other services processors to the right.

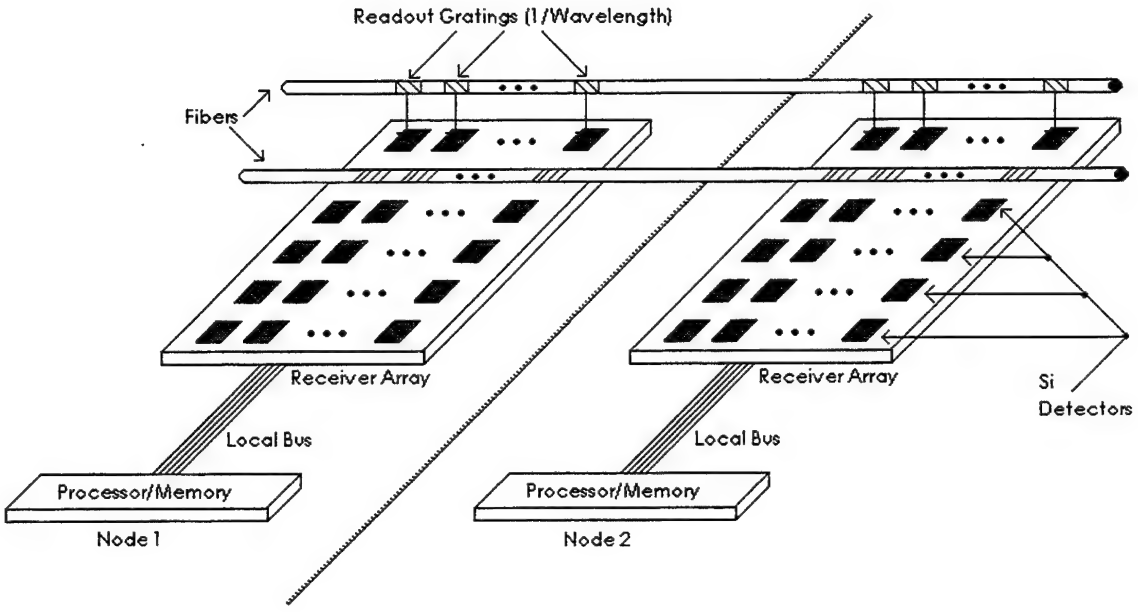


Figure 1.2 SOME-Bus Receiver Array

Figure 1.2 illustrates the optical fibers, with readout gratings, and receiver array used by two nodes in the interconnection network [26]. The readout gratings, permanently induced in the core of an optical fiber by standard holographic techniques, allow light to be coupled off the fiber and onto the detector [28]. Wavelength selectivity and reflectivity of the gratings are

determined by grating length, spatial period, and process exposure time. Each grating couples out a fraction of the light from one wavelength in the fiber to a designated detector on the receiver. The *N* receivers required by a single processor are fabricated as a thin film of amorphous silicon structures, which serve as the optical detectors, constructed directly on the surface of a digital CMOS device. The low conductivity of the amorphous silicon layer eliminates the need for subsequent patterning, thereby producing receivers with the same yield and cost as the CMOS device itself. Together with the transmitter design, this feature allows the SOME-Bus to scale by *O(N)*, minimizing both the number of expensive components (transmitters) and the cost of highly replicated components (receivers).

The current receiver-array performs two primary functions; it serves as the optical interface and as the processor interface [28]. The optical interface includes physical signaling, address filtering, barrier synchronization, length monitoring, and type decoding. When a detector receives a signal, it generates a bit stream which is examined for the presence of the framing byte. This is necessary because the start of a data byte occurs asynchronously. Bit stuffing at the transmitter and destuffing at the receiver is used to insure that the framing byte cannot occur within the packet. Destuff logic generates a byte stream, representing packet headers and data, and provides a framing error indication. Header decoding identifies message type (synchronization or data), destination address and message length. Synchronization messages are handled by barrier logic; for data messages, the destination address is compared to the set of valid addresses contained in the address decoder. The address decoder can

recognize individual, multicast-group, and broadcast addresses. If the address does not match, the message is ignored.

For data messages, once a valid address has been identified, the processor interface routes and queues the message. With one queue per channel, a node can receive an arbitrary number of messages simultaneously. Each queue is large enough to contain a single full-size ethernet packet or a number of smaller messages. The next stage of the interface determines which queue will be processed next. If most queues contain messages most of the time, a simple counter design selects the queues in a fair manner. If only a few queues contain messages at any time, then a resolver network makes the selection based on an indexing and priority scheme. The selected message is transferred directly into the processor's memory using cut-through routing hardware.

In the SOME-Bus, where each processor can broadcast its own message while simultaneously receiving messages from all other processors, synchronization is possible in a single cycle. To achieve this, the receiver array converts each synchronization message into a single bit of information and stores it in a flip-flop. The flip-flop output becomes one input to an and-tree consisting of partition members. This scheme limits the involvement of the processor in a synchronization operation to transmission of the synchronization message, testing the summary from the receiver array. Arbitrary group partitioning for synchronization is handled by special hardware. Each receiver has two counters, one tracking the number of partitioning decisions made by the receiver array's processor, the other tracking partitioning decisions of the

input channel. These counters tabulate the (respective) number of partition decisions since their last shared partitioning. When both counters are at zero, both processors are in the same partition. If they are not equal to zero, the respective processor(s) must undo the counted number of partition decisions to again be in the same partition.

To illustrate the data transfer capability of the SOME-Bus, a 128 node system will serve as an example. Using the silicon receiver array design described above and 32 fibers multiplexing 17 wavelengths/fiber (4 wavelengths/channel, 4 channels/fiber, 1 common clock) provides 77 MB/sec of bandwidth for each channel, given a clock rate of 155MHz [26]. Using a receiver array manufactured using gallium arsenide lift-off technology and a clock rate of 1 GHz, the same 32-fiber SOME-Bus would provide up to 500 MB/sec of bandwidth per node, equivalent to the link bandwidth of a Cray T3E [49].

In summary, the SOME-Bus features high bandwidth (scaling directly with the number of nodes), low latency, no arbitration delay, and non-blocking communication. Nodes communicate through a wavelength-division-multiplexed, optical-interconnection network implemented with laser-diode transmitters, optical-fiber transmission lines with induced Bragg gratings, and receivers consisting of CMOS devices with integrated amorphous silicon detector superstructures. The network supports individual, multicast, and broadcast transmissions, is (effectively) fully connected, and scales by $O(N)$.

1.3 Evaluation of Performance Potential

Determining the performance potential of the SOME-Bus requires comparing its network characteristics to those of existing systems. As explained at the beginning of this chapter, the method chosen for the comparison involves separating the networks represented in Bell's taxonomy [4], [22] into static and dynamic topologies. The quantifiable characteristics associated with each type are then compared to the SOME-Bus.

Table 1.1 Static Network Performance Comparison

NETWORK	*DIAMETER*	*NODE DEGREE*	*BISECTION*	*COST*
SOME-Bus	1	N	N	$N/4$
Completely-Connected	1	N-1	$(N/2)^2$	N-1
2D Mesh	$2(N^{1/2}-1)$	2	$N^{1/2}$	$2(N-N^{1/2})$
2D Torus	$2\lfloor N^{1/2}/2 \rfloor$	4	$2N^{1/2}$	$2N$
Hypercube	$\log_2 N$	$\log_2 N$	$N/2$	$N \log_2 N/2$

Performance metrics of static interconnection networks include diameter, connectivity, bisection width, and cost [22], [29]. Diameter represents the maximum number of communication links between any two processors in the system; a smaller diameter results in lower data-transfer latency. Connectivity, also referred to as node degree, describes the minimum number of communication channels that must be removed to separate a single network into two disconnected parts. Higher connectivity results in less contention for

shared communication resources. Bisection width is a measure of the minimum number of communication links that must be removed to separate a network into two equal halves. The greater the number of links, the more bandwidth for data transfer. The final criterion, cost, denotes the number of physical communication links in the system. Table 1.1 compares an *N*-node SOME-Bus to *N*-node static network topologies. As Table 1.1 shows, the SOME-Bus is clearly the superior network configuration among this group, matching or exceeding the best in every category except bisection width. Performance figures cited in Table 1.1, for systems other than SOME-Bus, came from the work of Bell [22] and Kumar [29].

Figures of merit for dynamic networks include data transfer latency, bandwidth per processor, wiring complexity, and switching complexity [22]. Latency refers to the minimum amount of time necessary to transfer data from source to destination. Bandwidth describes the total bandwidth available for data transfer by a single node. Wiring complexity measures the physical number of wires (fibers) interconnecting source and destination. Switching complexity defines the number of switches in a system.

Table 1.2 Dynamic Network Performance Comparison

Network	*Latency*	*Bandwidth*	*Wiring Complexity*	*Switching Complexity*
SOME-Bus	Constant	$O(wN)$	$O(N/w)$	$O(1)$
Bus	Constant	$O(w/N)$ to $O(w)$	$O(w)$	$O(N)$
Multistage	$O(\log_k N)$	$O(w)$ to $O(wN)$	$O(Nw\log_k N)$	$O(N\log_k N)$
Crossbar	Constant	$O(w)$ to $O(wN)$	$O(N^2w)$	$O(N^2)$

Table 1.2 compares the three dynamic network topologies (bus, multistage, and crossbar) represented in Bell's taxonomy to the same SOME-Bus configuration used in the static network comparison. Each system has *N* processors and memory modules and a data-channel width of *w* bits. In addition, the multistage network is assumed to use *k* x *k* switches. Based on the results shown in Table 1.2, it appears the SOME-Bus also has greater performance potential than existing dynamic topologies. An additional benefit, not explicitly seen in Table 1.2, is that SOME-Bus transmissions are never blocked due to contention for shared switching resources.

1.3.1 Optically-Based Networks

In addition to the static and dynamic networks above, the potential of a SOME-Bus must be compared to other optically-based parallel processors. In one research study, free-space optical techniques are applied to the design of a mesh-connected bus network [33]. The drawbacks to this design, compared to the SOME-Bus, include contention resolution due to multi-access communication channels and greater power requirements resulting from free-space transmission methods. A second study describes an optical crossbar design using time division multiplexing [43]. Although the optical crossbar switching network used to implement the crossbar reduces the switching complexity, allowing linear scaling of switching components, the contention for shared resources inherent in any crossbar system remains.

In two separate studies, systems were found that leverage optical transmission methods to implement a hyperbus (also known as a hypermesh).

This network topology provides complete connectivity in each dimension and appears isomorphic to a 1-D SOME-Bus. The concept of a hyperbus has been known for some time and shown to have great performance potential. However, it has not been seriously pursued until recently because its implementation is not feasible using electrical interconnection methods alone. In the first study, an implementation using a combination of electrical and optical crossbars is described [53]. Although the design uses multiple wavelengths per fiber to keep cost low, multiple-access of those wavelengths requires the additional complexity of contention resolution. The second proposal is based on completely-connected, 36-node building blocks [58]. Each node consists of a 'cluster' of 8 processors interconnected by a crossbar switch. Nodes within a building block are then completely connected via a clear plastic bar acting as an optical waveguide. Mirrors guide multiple wavelength-division-multiplexed optical signals through the bar to individual wavelength detectors at each node. Although promising as a feasible way to implement a hypermesh, this architecture has much greater complexity than SOME-Bus. In addition to the crossbar switch at each node, it requires more transmitters and precise mirror-to-detector alignment.

In summary, the SOME-Bus exhibits outstanding performance potential. Compared with static topologies, a SOME-Bus offers the same performance as the powerful but costly fully-connected network, but with the added benefit of a lower cost than a hypercube, mesh, or torus. Contrasted to the most versatile dynamic network, the crossbar, SOME-Bus scales better and supports more communication patterns. In addition, due to the optical implementation of its

broadcast-based design, no node is ever blocked from transmitting by another node, and no arbitration for shared resources is required. Even with the SOME-Bus requiring N^2 receivers, larger than the number required in other architectures, it scales by $O(N)$, since N receivers are integrated into a single device at relatively low cost.

1.4 Parallel Systems, Workload and Performance

Parallel systems can be classified as multicomputers, or multiprocessors [4], [22]. Multicomputer systems consist of multiple autonomous computing nodes, each with a distinct address space. All communication in multicomputer systems takes place through the interconnection network via message passing. Although existing multicomputer systems use both static and dynamic networks, static topologies are more common.

Multiprocessor systems are defined by a single address space, implemented as either centralized or distributed memory. In all multiprocessor systems, interprocessor communication takes place using shared variables in the common address space. In a central-shared-memory (CSM) multiprocessor, main memory is realized by a subsystem that is physically separate from the processing nodes. Since all main-memory requests traverse the interconnection network, CSM systems feature uniform access time to any memory address. The majority of CSM systems are implemented using dynamic interconnection networks, and are often referred to as tightly-coupled due to extensive sharing of common resources.

Distributed-shared-memory (DSM) systems have main memory physically distributed among the processing nodes. This distribution results in non-uniform access times since some main-memory requests are satisfied locally. Both static and dynamic networks are common in DSM system implementations. In these loosely-coupled multiprocessors, management agents map the shared logical address space onto local memories [24]. These agents hide the message-passing mechanism, provide a shared-memory model, and keep data coherent at all times. In addition, on each access to shared space, hardware determines if the requested data is in local memory and, if not, copies it from remote memory.

Recent trends in commercial parallel-system design indicate a preference for DSM multiprocessor systems [50]. This shift stems from two primary factors: first, given certain interconnection topologies DSM multiprocessors provide the economic advantages of both size- and generation-scalability by leveraging commercial microprocessor advances; second, their single address space allows for relatively efficient general-purpose programming. In addition, message passing can be implemented implicitly on DSM systems via writing to shared variables.

Given this preference for DSM multiprocessors, it is important to determine the issues that effect successful DSM systems and see if they exist in the SOME-Bus. A focus of current DSM research involves attempts to reduce or hide data access latency while maintaining memory consistency. In general, the result of this research indicates that data access latency in DSM systems is related to the extent of memory consistency required [24]. Models with strong restrictions show

increased latency and higher network bandwidth requirements (stronger restrictions result in greater traffic on the network). Models with weaker constraints that allow reordering, pipelining, and overlapping of memory produce better performance, but require explicit synchronization operations. This body of research implies that a viable DSM system should have a low-latency, high bisection-bandwidth interconnection network. These are characteristics the SOME-Bus possesses.

The following subsections first define the types of communication patterns that frequently occur in parallel applications, and then examine current research into their effect on system performance. Using the trend toward DSM systems as a guide, the research summaries are divided into those covering message-passing and central-shared-memory systems and those dealing with distributed-shared-memory systems. These studies show that today's high-performance computing workload includes modeling and simulation of physical phenomena, integrated circuits, neural networks, weather, economic systems, and image processing. These same studies show that processing these tasks on existing parallel systems results in load imbalance among processing nodes, delays caused by barrier synchronization, and communication patterns which place an excessive load on the interconnection network.

Some of the basic mathematical operations used to execute the workload described above include fast Fourier transforms (FFTs), matrix multiplication, Gaussian elimination, LU-factorization, and solutions to partial differential equations [29]. Parallel-processing implementations of these operations typically involve one-to-all and all-to-all broadcasts, and broadcasts

place the heaviest burden on inter-processor communication networks. The research studies will highlight the mismatch between these communication patterns and current interconnection architectures (both static and dynamic), regardless of implementation medium. The cumulative results of these studies indicate that high-performance applications execute on existing parallel systems at only poor-to-moderate performance levels, even after extensive efforts at software tuning. Based on these findings, it is apparent that a scaleable parallel system, capable of supporting the communication patterns inherent in common applications, is needed.

1.4.1 Communication Patterns

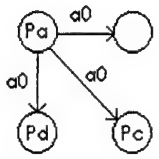
Efficient communication is critical to any application's performance. Communication may involve point-to-point operations, which have a single source/destination pair, or collective operations, in which more than two processes participate [21]. A collective operation begins when a group, consisting of some or all processes in an application, invokes a communication routine, such as a broadcast. The broadcast routine identifies the scope of group membership to determine if the broadcast will be implemented as a multicast (when the group does not consist of all processes in the application), or as a true broadcast. The range of collective operations, illustrated in Figure 1.3 [21] for a group of four processes, can be classified into three types: data movement, process control, and global compute operations.

The first type of collective operation, data-movement, is shown in Figure 1.3(a)-(e). Figure 1.3(a) shows a one-to-all broadcast in which one process (*Pa*)

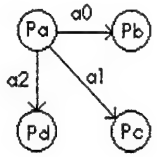
sends all other group members the same information (*a0*). A scatter operation, Figure 1.3(b), occurs when one process in the group (*Pa*) sends individual messages (*a0, a1, a2*) to other members. During a gather, illustrated in Figure 1.3(c), one process (*Pa*) is the recipient of information (*b0, c0, d0*) from other group members. In an all-to-all broadcast each group member (*Pa, Pb, Pc, Pd*) sends all other members an identical message (a0, *b0, c0, d0*) as shown in Figure 1.3(d). The final data-movement operation, an all-to-all scatter-gather, also known as an all-to-all personalized broadcast or a complete exchange, places the heaviest burden on a communication network [29]. Figure 1.3(e) shows how each process in the group (*Pi*) sends individualized information (*i0, i1, i2*) to all other group processes during this operation.

Barrier synchronization is an example of process control, the second category of collective communication,. As seen in Figure 1.3(f), one method of performing barrier synchronization occurs in two phases, with one process in the group (*Pa*) playing the role of a barrier process. In the first phase each member of the group that reaches the barrier sends a message indicating this fact to the barrier process. Once messages from all the other members are received, phase two takes place as the barrier process broadcasts a message to the group indicating that work may proceed.

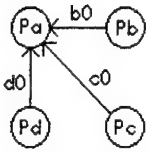
The final category of collective communication, global compute operations, includes both reduction and scan (also known as parallel prefix or prefix sum). In reduction, each member of the group forwards data to a central process (*Pa*) where associative and commutative operations are performed; examples include sum, max, min, and bitwise operations. Figure 1.3(g) shows a



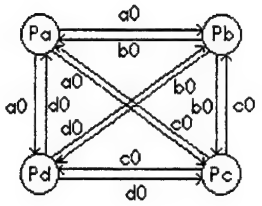
(a) Broadcast



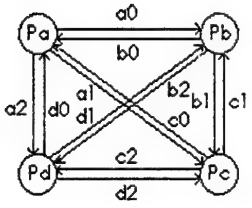
(b) Scatter



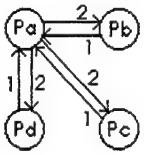
(c) Gather



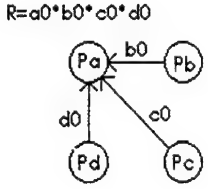
(d) All-to-All Broadcast



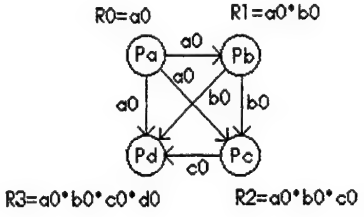
(e) All-to-All Scatter-Gather



(f) Barrier Synchronization



(g) Reduction



(h) Scan

Figure 1.3 Collective Communication Patterns

special case of reduction with a generic operator; the result *(R)* resides at a single process in this illustration while in other cases it may be distributed to some or all of the group processes. The scan operation, shown in Figure 1.3(h), applies an associative and commutative operator (represented by the asterisk) to the data such that the result at process *Pi* is *Ri = a0 * b0 ... * i0*.

1.4.2 Message-Passing and Central-Shared-Memory System Performance

One study that illustrates the effect of collective communication on message-passing or central-shared-memory system performance examines a parallel three-dimensional Navier-Stokes solver. This study analyzed the communication patterns inherent in the algorithm and evaluated its performance on *P*-processor IBM SP2, Cray T3D, and SGI Power Challenge XL systems [14]. The algorithm, based on 2- and 3-D FFTs, requires a series of all-to-all scatter-gather (complete exchange) operations, global reduction, gather operations, and global synchronizations. It was found that there are not enough network connections between nodes in any of these systems to support an *O(P)* complete exchange without contention, and that there was a large variability for times associated with communication performance, all biased toward higher values. Although the program is computation-intensive, memory-bus contention, brought on by the collective operations cited, caused excessive performance degradation in the Power Challenge. Performance of the SP2 and T3D was also degraded, but to a lesser extent. As the number of processors was increased, performance became increasingly communication-bound with the cost of the

complete exchange almost doubling (thus becoming prohibitively expensive) on the SP2.

Another study focused on the parallel implementation of an atmospheric general-circulation model making extensive use of Fourier transforms [37]. The application was executed on Intel Paragon and Cray T3D systems, where the authors found it necessary to perform data-shuffling (which utilizes all-to-all broadcast) to achieve load balancing among the processors. Modifications were made to the model in an effort to minimize this costly operation, but came at the expense of other forms of collective communication coupled with a substantial amount of local bookkeeping. Despite the modifications, processor utilization (a typical performance metric representing the percentage of processor time dedicated to problem solution) of only 30% to 40% was achieved.

Research involving the simulation of an astrophysics N-body problem using mature code compared the performance of two experimental parallel computers built with commodity (commercially available microprocessor) components to that of existing supercomputers that had been running the same code for years [55]. Although the code has been optimized, it generates numerous requests for non-local data yielding high communication costs. This study highlights the relationship between high communication costs and scalability of existing systems since all the computers with more than 100 processors attained only 30% processor utilization except an Intel Delta, which reached 50%.

The goal of one study was to reduce the cost of all-to-all communication (given all messages of the same size) in a 2-dimensional torus of size ($n \times n$) [19].

The author explains that this communication pattern is frequently encountered in multidimensional convolution, array transposes, etc., and often results from High-Performance FORTRAN operations. Presenting a communication algorithm based on phased communications, where each phase approaches the peak bandwidth available in this architecture, a 40% reduction in the time to perform a 512 x 512 FFT is achieved. In addition, given bidirectional communication links, an all-to-all communication takes only $(n/2)^3$ phases using the proposed algorithm. Although the pattern seems optimal, the phases must be completely and globally synchronized through either more interprocessor communication or additional hardware, a costly addition in either case. It was also shown that performance deteriorates when message sizes vary within a phase, at its worst when empty messages are included.

Many-body simulation, a common high-performance application, is one in which arriving at a solution usually, sometimes exclusively, depends on all-to-all communication [17]. This study compares the performance of a new many-body algorithm, designed specifically to avoid all-to-all communication, to two more traditional methods. All algorithms were executed on both an nCUBE2 and an Intel Paragon system. When running on systems of 128 or more nodes, speedup (a ratio of execution time on the parallel system to that of a uni-processor system) was 50% or less of perfect (linear), with the rate of Paragon performance degradation greater than that of the nCUBE2 as more nodes were employed. In some situations the traditional methods, using all-to-all communication, performed better. A similar lack of scalability was found in a separate study of code designed for transient dynamic simulations, also

executed on a Paragon system [45]. Only 30% (64 nodes) to 47% (128 nodes) of linear speedup was reported by this study.

Utilizing both theoretical and simulation models, one study examined the performance of a clustered CSM multiprocessor. Crossbar switches connected processor elements (consisting of a processor and private memory) within a cluster, and those same processor elements with global memory [38]. As the probability of memory access/cycle was increased, the authors found that if all those accesses were directed to global memory, the system could only attain approximately 25% of its peak processing capability. However, performance rose to 60% when those same memory accesses were directed equally to private and global memory, thereby reducing the burden on the interconnection network.

There are numerous articles dealing with general issues involving collective communication in message-passing architectures that are not specifically tied to an application. There have been attempts to minimize the number of steps necessary to broadcast on hypercubes by using path-based algorithms [20], [23]. Similarly, there is a large body of research in multicast (broadcast to a select group of nodes) communication and its effect on mesh and torus architectures employing trees and wormhole routing [9], [30], [34], [40], [41]. One presentation noted that extensive efforts are focusing on the development of algorithms to alleviate the fact that intense multicast communications cause wormhole-routing performance to degrade to that of store-and-forward routing [29]. When viewed as a whole, the research above

indicates that collective communication inhibits the performance potential of existing message-passing and CSM systems.

1.4.3 Distributed-Shared-Memory System Performance

As with message-passing and CSM systems, there are numerous studies illustrating the impact of collective communication on the performance of existing DSM systems. One study uses both a queueing-network model and simulation to examine a two-dimensional mesh-based multiprocessor system where each processor can execute a number of threads [42]. Model parameters include number of threads per processor, probability that memory requests are directed to particular remote memories, and the delay in each network switch. Program execution is represented by each thread processing for a period of time, generating a memory request, and then suspending. The analysis of this model provides the data from which processor utilization and remote memory request response time is calculated. The results showed that when the probability of remote memory request was small, processor utilization increased as the number of threads executed increased, but response time (relative to mean thread run time) also increased dramatically. The data showed that response time becomes unacceptably large when more than five threads are present in each processor. Also, as the probability of remote memory requests increased (≥ 0.5), the interconnection network saturated and processor utilization fell below 35%.

In an article examining how to improve performance of the HP/Convex Exemplar system, an example of the role of the interconnection network in

application scalability is presented [51]. Processor performance is measured during execution of four Earth and space-sciences application programs. The Exemplar is a relative newcomer to the multiprocessing field, consisting of up to 16 multiprocessor clusters, each cluster containing four functional blocks, each block having two processors. The blocks within a cluster are interconnected by a crossbar switch, and sets of functional blocks between clusters are interconnected by four ring networks in an attempt to provide high connectivity at a relatively low cost. Using up to 16 processors, one portion of the study measured cache misses for one application (tree code for an N-body problem) finding the rate to be 10% to 25% depending on application size (verifying the need for an efficient interconnection network for these types of problems). A second part of the study, measuring the scalability of all the applications while varying the number of processors between 1 and 16, showed that some applications had almost linear speedup, while others fell below 50%. The study attributes poor performance to "irregular data access patterns, global communication between processors, and load balancing" for a finite element application and "irregular, dynamic data structures" for a particle-in-cell program. This study showed that the performance of one of the newest, most commercially-popular DSM systems suffers from poor network support of the collective-communication requirements inherent in commonly executed, high-performance applications.

Findings similar to those of the study above are seen in a study of four architectures with hardware DSM support [16]. After establishing workload parameters by estimation, an examination of the impact of system size and data

locality (both temporal and spatial) on processor utilization and shared-memory latency was undertaken. With workload parameters that appear overly optimistic, especially in terms of cache misses (as evidenced by the low network utilization in all four architectures here compared to those seen in other studies), the results show significant data-access latency compared to the CPU clock cycle time. In conjunction with that finding, data locality was found to have a significant effect on processor utilization.

Examining a DSM implementation of a system originally designed as a message-passing multicomputer is the topic of another study [2]. Using an nCUBE, a hypercube topology capable of supporting up to 8192 nodes, the study experiments with executing four parallel programs on a 16-node configuration. Although linear (or better) speedup was achieved in three applications, greatly reduced performance occurred in the fourth. This application, which performs matrix addition on distributed data, required significant data-transfer time relative to node computation time. The authors concluded that such programs are unsuitable for DSM implementation on a hypercube unless an algorithm can be found to reduce the communication involved. Taking a broader view, these results indicate that large applications distributed over many nodes (more than 16) require significant data-transfer times, and, since they cannot simply be rejected as unsuitable for DSM implementation, alternative networks must be developed that offer better support.

The fact that poor performance can occur even in a parallel system specifically designed to reduce contention for shared resources is found in

another study [5]. Using a model of a DSM multiprocessor with a multistage bus network, the authors study processor utilization. Model parameters include the probability of a processor generating a memory request during a cycle and the probability that a request is directed to local, versus non-local, memory. Results showed that when the majority of requests are directed to local memory and the probability of generating a memory request is above 10%, processor utilization falls below 65%. When the majority of requests are to non-local memory, and the probability of memory reference rises above 10%, utilization falls below 40%. Utilization drops significantly as the request probability increases, which the authors attribute to "a higher amount of traffic and queuing delays".

In summary, existing multicomputers and multiprocessors, implemented with static or dynamic interconnection networks, offer only poor-to-moderate performance for a wide class of commonly executed applications that use collective communication operations. This finding, combined with the results of the performance-potential evaluation, justify an in-depth performance study of the SOME-Bus architecture.

1.5 Outline of the Dissertation

Chapter 2 presents the development of a closed-queueing-network model of the SOME-Bus as a message-passing system. Chapter 3 validates a message-passing SOME-Bus simulator using this model, extends the simulator to perform synchronization tasks, and compares its results to message-passing crossbar and torus simulations.

In Chapter 4 two distributed-shared-memory models of the SOME-Bus are developed using queueing theory. Chapter 5 presents the results from both models and validates a corresponding simulator. That simulator is extended to perform cache coherence operations, and its performance is compared to DSM torus and crossbar simulations.

Chapter 6 offers conclusions based on the findings of this dissertation and describes the direction of future research.

Chapter 2

A MESSAGE PASSING MODEL OF THE SOME-BUS

2.1 Message-Passing Model

It is important to choose an analytical modeling method with a firm theoretical foundation to both evaluate performance and validate simulations. The model must abstract the system to a degree appropriate for evaluating changes in system parameters and offer an efficient solution method to facilitate those evaluations. Modeling the SOME-Bus as a closed queueing network satisfies all of these requirements.

Queueing network theory is based on the theory of Markov processes, a subclass of stochastic processes. A characteristic of Markov processes is that the probability of being in a specific next state depends solely on the current state, i.e., the evolution of the process is independent of its past history [27]. The process of interest for SOME-Bus evaluation is the distribution of messages among system resources. Since messages and system resources are discrete, finite entities, the state space defining their distribution is also finite and discrete, and the process is referred to as a Markov chain.

The closed queueing network model of a message-passing SOME-Bus system is illustrated in Figure 2.1. The model consists of a network of service centers that represent system resources, with each node modeled as two service centers in tandem. The first service center represents the activities of the receiver-buffer and processor-memory blocks of the SOME-Bus architecture, as

depicted in Figure 1.1, and will be referred to as the processor, or p-type, service center. The second service center models the actions of the transmitter block and the channel characteristics, and will be referred to as the channel, or c-type service center. These service centers are characterized via assignment of service rates and service disciplines. All service centers in the model of Figure 2.1 are assumed to have load-independent, exponential service time distributions, and first-come, first-served service disciplines.

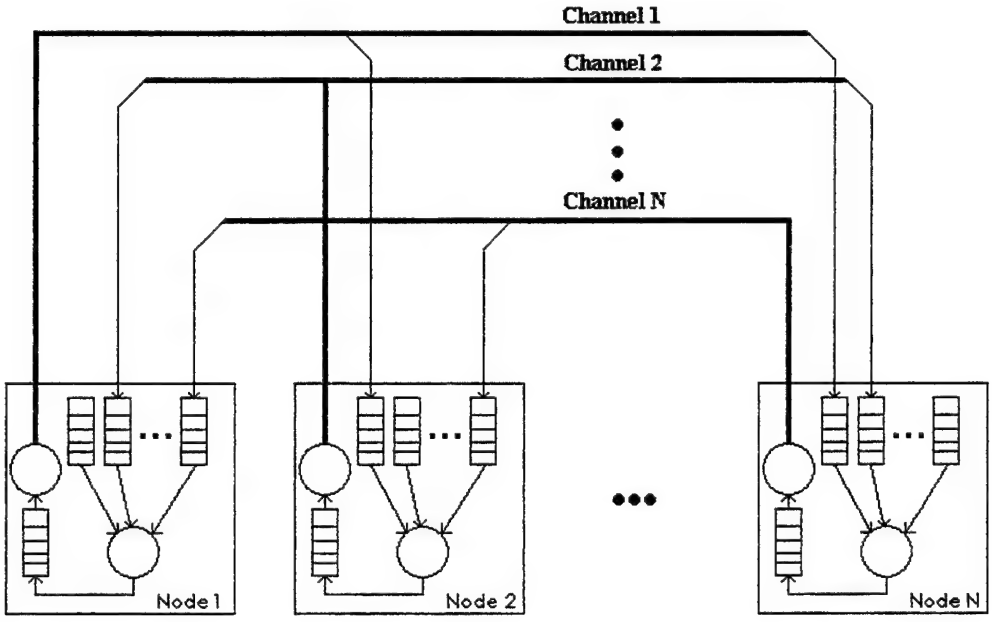


Figure 2.1 Queueing Network Model of an N-Node SOME-Bus

In a message-passing system, a program is viewed as a set of processes, and messages are used to exchange data between those processes [29]. Assuming each node is responsible for executing a fixed number of processes, and each process requires a data message to enable its execution, the number

of messages in the system is constant and equal to the total number of processes (*K*). No messages enter the system from external sources, or depart to external sinks. Also, since the same average service rate applies to all messages arriving at a given service center, a single-class closed queueing network results. In a single-class network the message originator becomes irrelevant, allowing the set of queues in each processor service center (as seen in Figure 2.1) to be replaced by a single queue, resulting in the simplified SOME-Bus model shown in Figure 2.2.

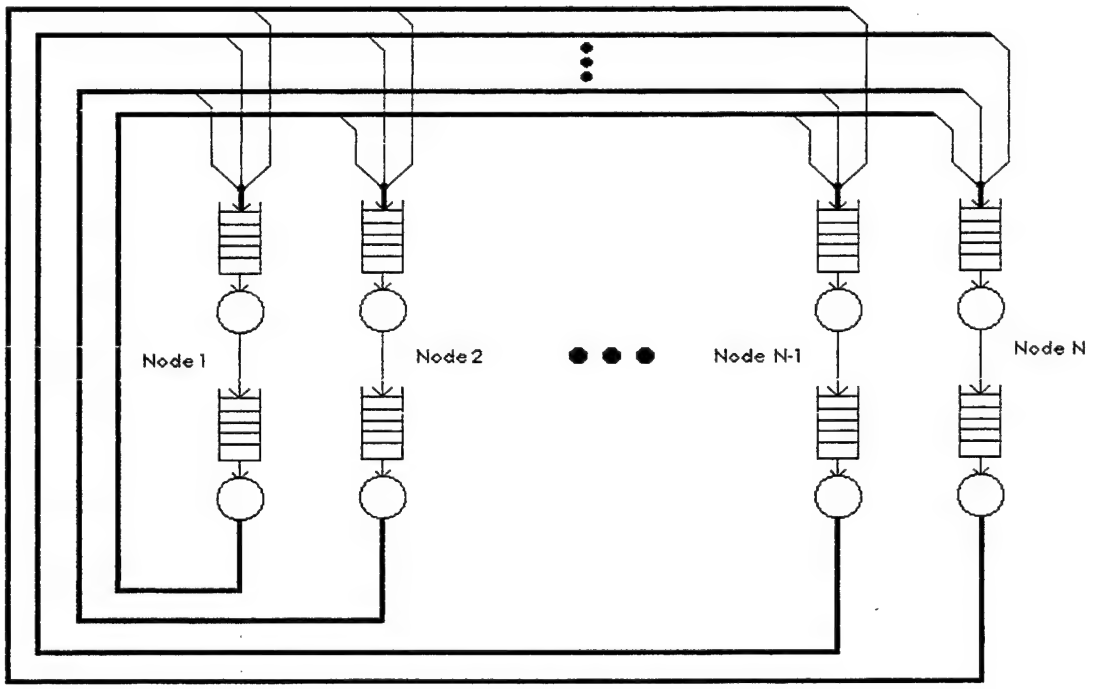


Figure 2.2 A Simplified N-Node SOME-Bus Model

A queueing network model of a Markov chain affords a closed-form equation, known as the product-form equation, for calculating equilibrium state probabilities [15], [39]. Once this equation is solved such that the sum of those

probabilities equals one, performance measures can be determined [32]. The product-form equation defining the state probabilities associated with the SOME-Bus model of Figure 2.2 appears as Equation 2.1, where *N* represents the number of nodes in the system, and k_i the number of messages at service center *i*.

$$P(k_1,k_2,...k_{2N}) = \left(1/C(K)\right) \cdot \left(\prod_{i=1}^{2N} h_i(k_i)\right) \qquad (\text{where } \sum_{i=1}^{2N} k_i = K) \qquad (2.1)$$

In Equation 2.1, $h_i(k_i)$ represents the individual, unnormalized queue-length distributions of all service centers in the network, which appear to act independently at equilibrium. The purpose of the normalization constant, *C(K)*, is to insure that the sum of the state probabilities equals one. Although a solution to Equation 2.1 exists, direct computation is inefficient, making performance evaluation of alternative system configurations cumbersome.

When service centers in a closed queueing network have exponential service time distributions, as assumed above, their queue-length distributions, $h_i(k_i)$, depend on their service and arrival rates. Service rates are simply the inverse of service times. The arrival rates, λ_j, are found by solving the system of flow-balance equations seen in Equation 2.2, where λ_i is the departure rate, and x_{ij}, is the fixed probability of a message transitioning from center *i* to center *j*.

$$\lambda_j = \sum_{i=1}^{2N} \lambda_i x_{ij} \qquad (2.2)$$

Letting **X** represent the matrix of transition probabilities and substituting λ_i for λ_j (since arrival and departure rates from service center *i* are the same at equilibrium), allows the system of equations, represented by Equation 2.2, to be written in the vector-matrix form $\boldsymbol{\lambda}=\boldsymbol{\lambda}\mathbf{X}$. Because this is a closed system, with equal arrival and departure rates, the system $\boldsymbol{\lambda}=\boldsymbol{\lambda}\mathbf{X}$ has one linearly dependent equation [27]. One method of solving this system is to assign one service center as a reference center, and let its arrival rate serve as a parameter. Using this method establishes relative arrival rates for all other centers [39], [54].

Assigning the load-independent service rate μ_i to service center *i* in Figure 2.2, and substituting the resulting unnormalized service center queue-length distribution, $(\lambda_i/\mu_i)^{k_i}$ for $h_i(k_i)$ in Equation 2.1 [48], results in Equation 2.3 (given the service time and service discipline assumptions already stated).

$$P(k_1,k_2,\ldots k_{2N}) = \left(1/C(K)\right)\cdot\left[\prod_{i=1}^{2N}\left(\lambda_i/\mu_i\right)^{k_i}\right] \qquad \left(\text{where } \sum_{i=1}^{2N} k_i = K\right) \tag{2.3}$$

The normalization constant, *C(K)*, is then found using Equation 2.4.

$$\sum_{\sum_{i=1}^{2N} k_i = K} P(k_1,\ldots,k_{2N}) = 1 \;\Rightarrow\; C(K) = \sum_{\sum_{i=1}^{2N} k_i = K}\left[\prod_{i=1}^{2N}\left(\lambda_i/\mu_i\right)^{k_i}\right] \tag{2.4}$$

Normalization constants are used in the computation of performance measures including server utilization (the proportion of time the server is busy), residence time (the average time spent at the service center by a customer, both queued and receiving service), queue length (the average number of customers at the service center, both waiting and receiving service), and throughput (the rate at which customers pass through the service center).

An efficient solution for Equation 2.4, given identical service rates μ_p and μ_c for all processor and channel service centers, respectively, has been found [25]. Transition rates were calculated by assuming that messages transmitted by c-type (channel) centers are uniformly directed to the p-type (processor) centers at other nodes. These assumptions established transition probabilities from channel to processor centers of $x_{ij} = 0$ (if the channel and processor centers are in the same node) or $x_{ij} = 1/(N-1)$ (otherwise). Since messages transferred from a p-type center can only go to the c-type center in the same node, transition probabilities from processor to channel are $x_{ij} = 1$ (if the channel and processor centers are in the same node) or $x_{ij} = 0$ (otherwise). There is a limitation to the existing solution; the solution is specific to the two-service-center model of a node shown in Figure 2.2. In addition, the solution depends on identical service and arrival rates at all nodes, prohibiting hot-spot analysis unless modifications are made. The next section presents the development of an efficient solution process that overcomes the two-service-center model limitation and allows evaluation of hot-spot conditions.

2.2 Calculating System Performance

As explained above, system performance measures can be calculated using network normalization constants [7]. The method developed in this dissertation for the determination of those constants is based on a service center aggregation technique known as Norton's Theorem for Queueing Networks (NTQN) [11]. Norton's theorem for queueing networks was originally developed as an efficient way to evaluate network performance while varying parameters of a subsystem-of-interest within a larger network. Standard application of NTQN involves functionally partitioning a network into a subsystem-of-interest, σ_1, and the remainder of the network, σ_2. A short replaces σ_1, and the resulting network is used to find a single, load-dependent (Norton-equivalent) aggregate center that exhibits the statistical behavior of σ_2, as experienced by σ_1. (The service rate for the equivalent service center is set equal to the load-dependent throughput of the shorted branch.) Once that equivalent service center is found, it replaces σ_2 in the original system to create a new network. The product-form equation for that network has fewer terms, making the computation of network normalization constants more efficient.

Evaluation of the SOME-Bus model focuses on how changing the service rate of processor or channel service centers impacts system performance, making a node the subsystem-of-interest. The fact that there are multiple nodes in the model requires a modification to the standard application of NTQN described above. First, a Norton-equivalent service center for the model of each node will be found, making individual nodes serve as σ_2 during the application of NTQN (with the remainder of the network serving as σ_1). The

node-equivalent service centers then replace all the corresponding subsystems (nodes) in Figure 2.2. Although the network models in this dissertation also incorporate the two-service-center model of a node used by Katsinis [25], this step allows the evaluation of any node-architecture model that meets NTQN requirements. Next, NTQN is iteratively applied to pairs of node-equivalent service centers until only an *N-1* node Norton's equivalent service center in series with a single node remains. The normalization constants for this equivalent network are used to calculate network performance.

2.2.1 Norton's Equivalent of a Node

The first step in finding network normalization constants is to determine a Norton's equivalent service center for each system node. Figure 2.3 depicts a single node, designated node *r* (for *reference*), serving as σ_2 with the remainder of the SOME-Bus network replaced by a short. Solving the resulting flow-balance equations, seen in Equation 2.5, shows that the arrival rate at the processor center, λ_{p_r}, is equal to the arrival rate at the channel center, λ_{c_r}.

$$\begin{bmatrix}\lambda_{p_r} & \lambda_{c_r}\end{bmatrix} = \begin{bmatrix}\lambda_{p_r} & \lambda_{c_r}\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \tag{2.5}$$

The resulting product-form equation for this closed queueing network appears as Equation 2.6. The number of messages in the processor and channel service centers are represented by k_p and k_c, and processor and channel service rates by μ_{p_r} and μ_{c_r} (respectively).

$$P(k_p, k_c) = \left(1/C_r(K)\right) \cdot \left(\lambda_{p_r}/\mu_{p_r}\right)^{k_p} \cdot \left(\lambda_{c_r}/\mu_{c_r}\right)^{k_c} \quad (\text{where } k_p + k_c = K) \tag{2.6}$$

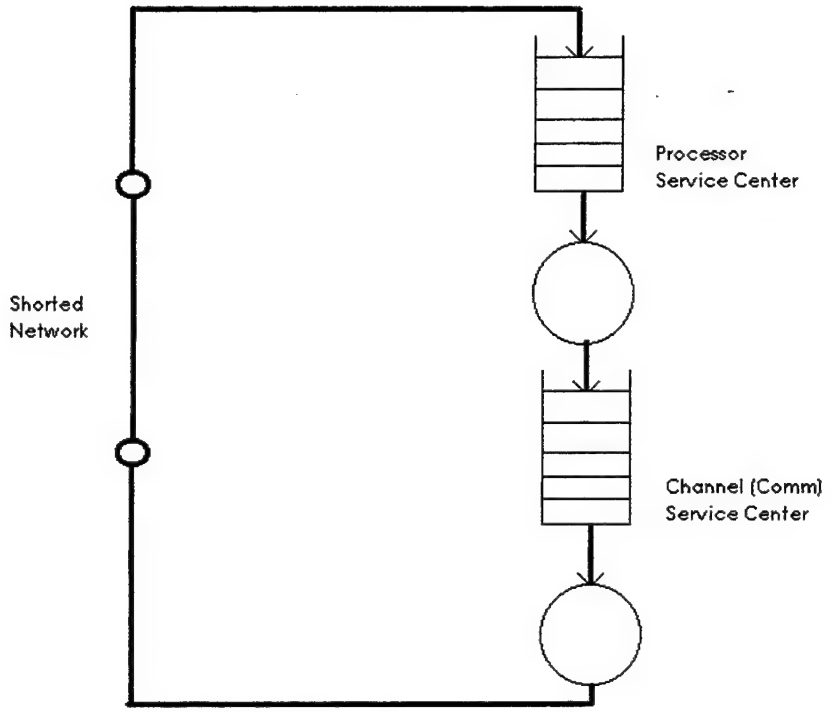


Figure 2.3 Finding the Norton's Equivalent Service Center for a Single Node

Choosing the arrival rate of the processor center to serve as the system parameter and arbitrarily assigning it the value of the processor service rate, μ_{p_r}, produces the normalization constant for node *r*, $C_r(K)$, seen in Equation 2.7,

$$C_r(K) = \sum_{k_c=0}^{K} \left(\mu_{p_r}/\mu_{c_r}\right)^{k_c} \tag{2.7}$$

which simplifies to

$$C_r(K) = \frac{\left[1 - \left(\mu_{p_r} / \mu_{c_r}\right)^{K+1}\right]}{\left[1 - \left(\mu_{p_r} / \mu_{c_r}\right)\right]} \tag{2.8}$$

Once the normalization constant is found, average throughput, $Y_r(K)$, for the processor and channel service centers in Figure 2.3 can be calculated using Equation 2.9 [7].

$$Y_i(K) = \lambda_i \cdot \left(C_r(K-1) / C_r(K)\right) \quad (i = p, c) \tag{2.9}$$

With the arrival rates of both service centers the same, the throughput of both centers, and the shorted branch, are equal. By calculating $C_r(k)$ for all k, k = $1,2,...,K$, and substituting the parameter value μ_{p_r} for λ_i, the solution to Equation 2.9 becomes the load-dependent (state-dependent) service rate for the Norton's equivalent of node r [48].

$$\mu_r(k) = \mu_{p_r} \cdot \left(C_r(k-1) / C_r(k)\right) \quad (k = 1,2,...,K) \tag{2.10}$$

Node-equivalent service centers must be found for every node with processor or channel service rates, or arrival rates, different from those of node r.

By assigning μ_{p_r} to serve as the parameter value representing the arrival rate to the reference node, relative arrival rates to the remaining nodes are found by solving $\lambda = \lambda \mathbf{X}$ for the *N*-node system. Normalization constants can then be calculated for node *i* using Equation 2.11,

$$C_i(K) = \sum_{j=0}^{K} \left[\left(\lambda_i / \mu_{p_i} \right)^j \cdot \left(\lambda_i / \mu_{c_i} \right)^{K-j} \right] \tag{2.11}$$

which results in the associated load-dependent service rate for node *i* seen in Equation 2.12.

$$\mu_i(k) = \lambda_i \cdot \left(C_i(k-1) / C_i(k) \right) \quad (k = 1,2,...,K;\ i = 1,2,...,N) \tag{2.12}$$

Substituting these Norton-equivalent service centers for their respective nodes in Figure 2.3 results in the simplified system model shown in Figure 2.4. The product-form equation for this network, with its *N* load-dependent service centers, takes the form of Equation 2.13 [15].

$$P(k_1, k_2, ... k_N) = \left(1 / C(K) \right) \cdot \left[\prod_{i=1}^{N} \left((\lambda_i)^{k_i} \Big/ \prod_{j=1}^{k_i} \mu_i(j) \right) \right] \quad \left(\text{where } \sum_{i=1}^{N} k_i = K \right) \tag{2.13}$$

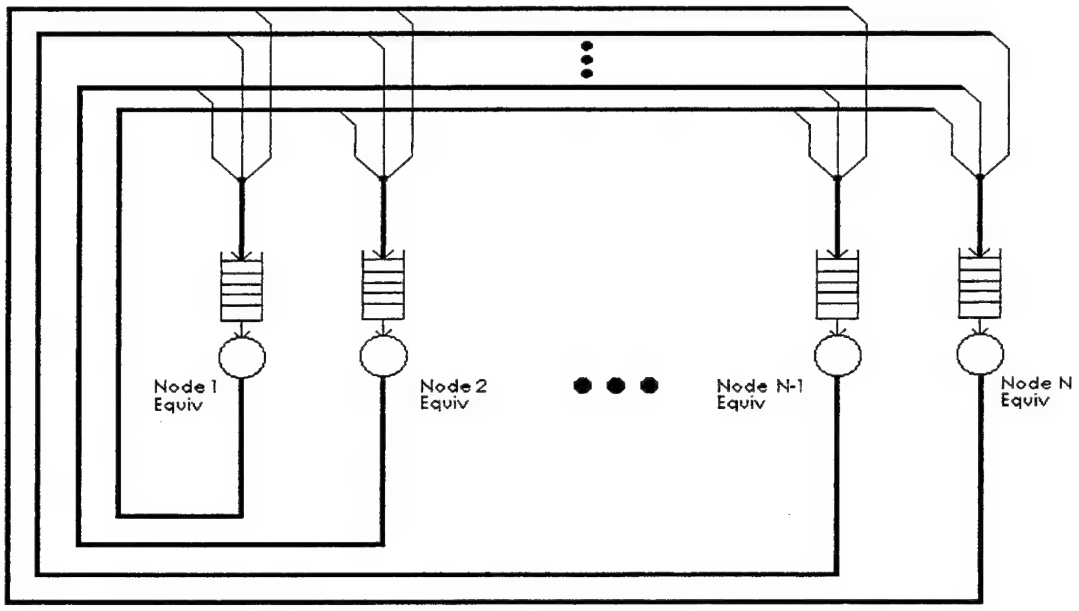


Figure 2.4 N-Node SOME-Bus Model w/ Norton's Equivalent Nodes

2.2.2 Network Normalization Constants

Determining network normalization constants first requires finding an *N-1* node Norton's equivalent service center, which is accomplished here by iterative application of NTQN. The process begins by partitioning the model of Figure 2.4 into two parts. One part, σ_2, consists of any two, node-equivalent service centers (identified as Center *a* and Center *b*), while σ_1 contains the remainder of the network. Shorting σ_1 results in the queueing network of Figure 2.5, allowing the Norton's equivalent center of σ_2 to be determined. This equivalent service center is substituted for σ_2 in the model of Figure 2.4, and the process is repeated until an *N-1* node aggregate center results.

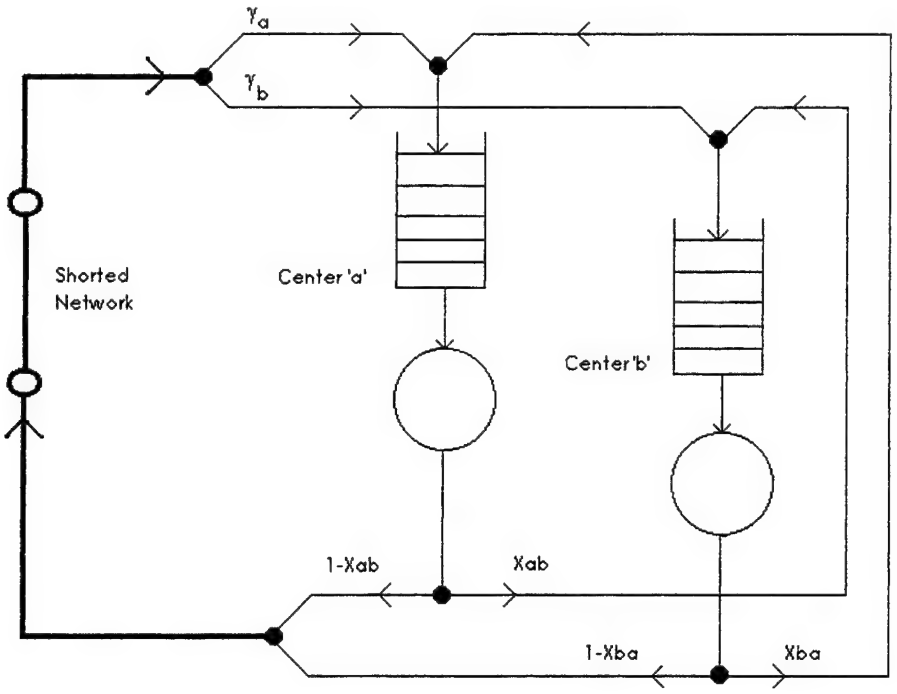


Figure 2.5 Model for Norton's Equivalent Reduction of Two Centers

Terms appearing on Figure 2.5 indicate the probability of a message traversing the associated branches of the network. The values x_{ab} and x_{ba} are the system transition probabilities from Node *a* to Node *b*, and Node *b* to Node *a* (respectively). Values γ_a and γ_b arise from system flow-balance considerations, and are calculated using system arrival rates and transition probabilities, as seen in Equation 2.14.

$$\gamma_a = \left(\sum_{\forall i \neq a,b} \lambda_i \cdot x_{ia}\right) \Big/ \left(\sum_{\forall i \neq a,b} \lambda_i \cdot (x_{ia} + x_{ib})\right) \quad \text{and} \quad \gamma_b = \left(\sum_{\forall i \neq a,b} \lambda_i \cdot x_{ib}\right) \Big/ \left(\sum_{\forall i \neq a,b} \lambda_i \cdot (x_{ia} + x_{ib})\right) \tag{2.14}$$

For reference purposes, the transition probability matrix for the network of Figure 2.5, is shown in Equation 2.15.

$$\mathbf{X} = \begin{bmatrix} x_{aa} & x_{ab} \\ x_{ba} & x_{bb} \end{bmatrix} \quad \text{where}$$

$$\begin{aligned} x_{aa} &= (1 - x_{ab}) \cdot \gamma_a \\ x_{ab} &= x_{ab} + (1 - x_{ab}) \cdot \gamma_b \\ x_{ba} &= x_{ba} + (1 - x_{ba}) \cdot \gamma_a \\ x_{bb} &= (1 - x_{ba}) \cdot \gamma_b \end{aligned} \tag{2.15}$$

The product-form equation for the network of Figure 2.5 is created using the load-dependent service rates and system arrival rates for Centers a and b [15].

$$P(k_a, k_b) = \left[1 / C_{\sigma_2}(K) \right] \cdot \left((\lambda_a)^{k_a} / \prod_{i=1}^{k_a} \mu_a(i) \right) \cdot \left((\lambda_b)^{k_b} / \prod_{j=1}^{k_b} \mu_b(j) \right) \quad (\text{where } k_a + k_b = K) \tag{2.16}$$

Substituting the node-equivalent, load-dependent service rate equation (Equation 2.12) for each center into Equation 2.16, and noting that $C(0) = 1$ by definition [48], the normalization constants for the network of Figure 2.5 are found.

$$C_{\sigma_2}(k) = \sum_{j=0}^{k}\left[\left(C_a(j)\right)\cdot\left(C_b(k-j)\right)\right] \qquad (k = 0,...,K) \tag{2.17}$$

The load dependent service rate for the two-center aggregate is set equal to the throughput of the shorted-network branch [48], as shown in Equation 2.18.

$$\mu_{\sigma_2}(k) = \left[\left(1-x_{ab}\right)\cdot\lambda_a + \left(1-x_{ba}\right)\cdot\lambda_b\right]\cdot\left[C_{\sigma_2}(k-1)\Big/C_{\sigma_2}(k)\right] \tag{2.18}$$

Comparing Equation 2.18 with Equation 2.9, the arrival rate of the aggregate service center can be determined.

$$\lambda_{\sigma_2} = \left(1-x_{ab}\right)\cdot\lambda_a + \left(1-x_{ba}\right)\cdot\lambda_b \tag{2.19}$$

The final step in the aggregation process consists of revising and reducing the transition probability matrix associated with the system model of Figure 2.4 to reflect the aggregation of two centers into one. Replacing the separate transition probabilities from node *i* to nodes *a* and *b* is the single transition probability, shown in Equation 2.20, from node *i* to the aggregate center.

$$x_{i\sigma_2} = x_{ia} + x_{ib} \qquad (i \neq a,b) \tag{2.20}$$

The two rows of the system transition probability matrix associated with Centers *a* and *b* are replaced by the transition probabilities from the aggregate center to other network nodes.

$$x_{\sigma_2 j} = \left(\lambda_a \cdot x_{aj} + \lambda_b \cdot x_{bj}\right) \Big/ \sum_{\forall j \neq a,b} \left(\lambda_a \cdot x_{aj} + \lambda_b \cdot x_{bj}\right) \qquad (j \neq a,b) \qquad (2.21)$$

The aggregate node does not send messages to itself, making $x_{\sigma_2 \sigma_2} = 0$.

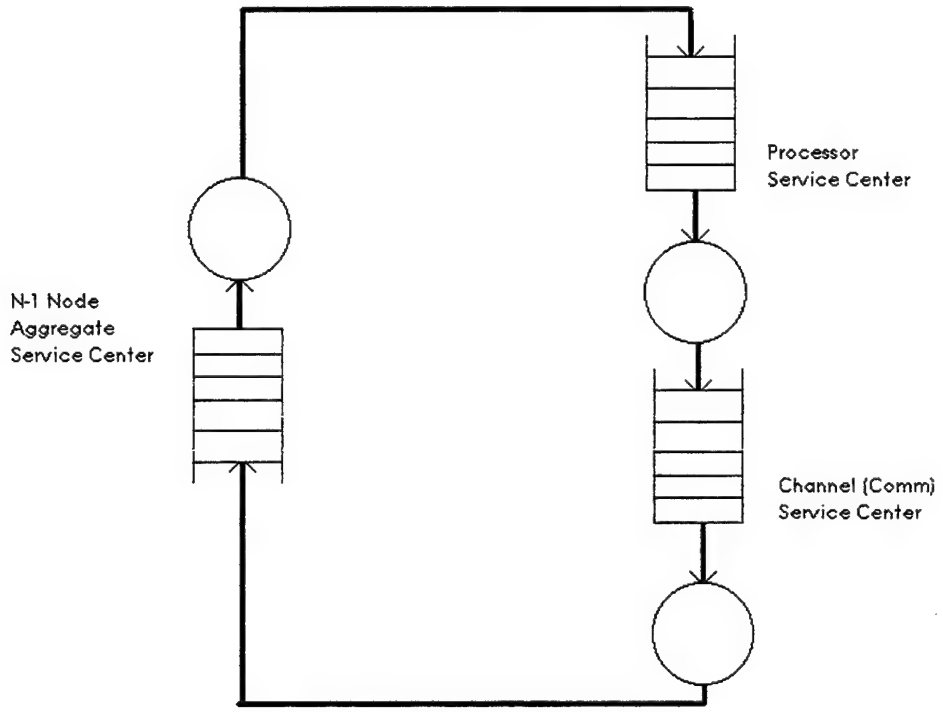


Figure 2.6 Reduced Model for Performance Evaluation

In a network with equal transition probabilities, such as a SOME-Bus with no hot-spots and identical processor and channel service rates at every node, repeating the aggregation process as few as $2[\log_2(N)]$ times produces an *N-1* node Norton's equivalent service center. The worst case, where no two pairs of nodes have the same transition probabilities requires *N-2* iterations.

The model representing the final result of the aggregation process just described is shown in Figure 2.6, with the original two-center model of the remaining node replacing its Norton equivalent for the purpose of performance evaluation. This model, statistically equivalent to the model of Figure 2.4, produces the network normalization constants used to calculate performance measures.

The product-form equation for Figure 2.6 appears as Equation 2.22. Due to the series configuration of the network, the arrival rates to all the stations are equal ($\lambda_p = \lambda_c = \lambda_{agg}$).

$$P(k_p,k_c,k_{agg}) = 1/C_{Net}(K)\cdot\left(\lambda_p/\mu_p\right)^{k_p}\cdot\left(\lambda_c/\mu_c\right)^{k_c}\cdot\left(\left(\lambda_{agg}\right)^{k_{agg}}\Big/\prod_{j=1}^{k_{agg}}\mu_{agg}(j)\right) \qquad (2.22)$$

$$(\text{where } k_p + k_c + k_{agg} = K)$$

For calculation of $C_{Net}(k)$, Equation 2.23 partitions the messages in the network into those in the two service centers of the node (*i*), and those in the aggregate service center (*k-i*). It then further partitions those in the node into those in the processor center (*j*) and those in the channel center (*i-j*).

$$C_{Net}(k) = \sum_{i=0}^{k}\left[\left[\sum_{j=o}^{i}\left(\lambda_p/\mu_p\right)^{j}\cdot\left(\lambda_c/\mu_c\right)^{i-j}\right]\cdot\left(\left(\lambda_{agg}\right)^{k-i}\Big/\prod_{j=1}^{k-i}\mu_{agg}(j)\right)\right] \quad (k = 0,1,\ldots,K) \quad (2.23)$$

This reduces to Equation 2.24.

$$C_{Net}(k) = \sum_{i=0}^{k}\left[\left[\sum_{j=o}^{i}\left(\lambda_p/\mu_p\right)^{j}\cdot\left(\lambda_c/\mu_c\right)^{i-j}\right]\cdot\left(C_{agg}(k-i)\right)\right] \quad (k = 0,1,\ldots,K) \quad (2.24)$$

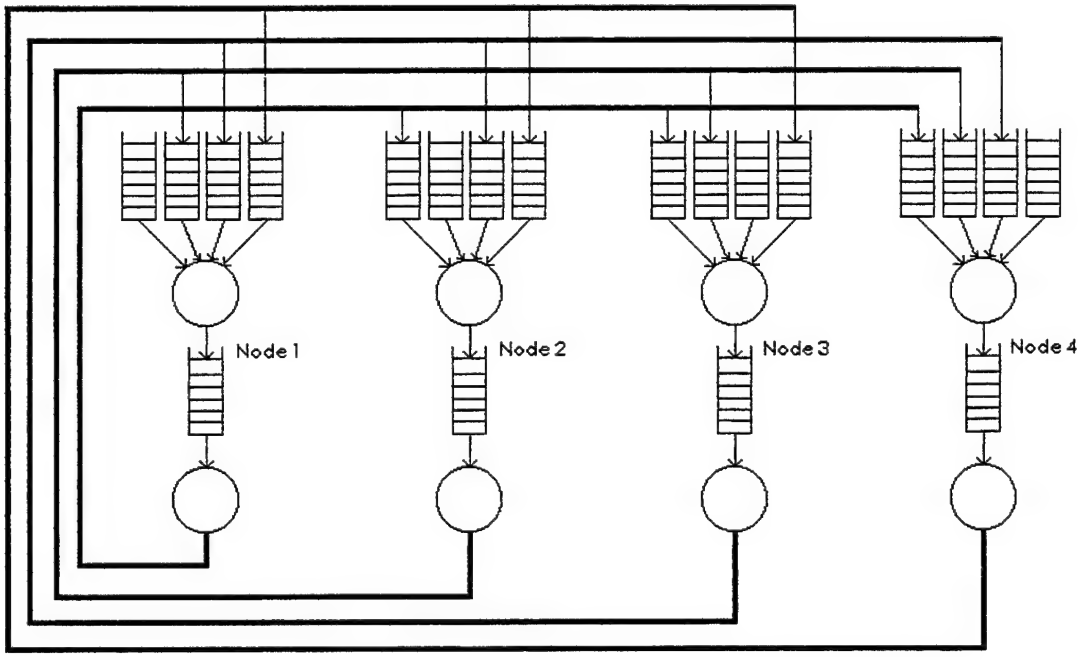


Figure 2.7 A Four-Node SOME-Bus System

The following subsections present three examples illustrating how to apply the method described above. The first example represents a SOME-Bus system with no hot-spots. The second system has a single hot-spot node that is the focus of the evaluation. The final example has the same hot-spot node, but a different node is the subject of the evaluation . All nodes are assumed to have identical processor and channel service rates. Figure 2.7 illustrates the queueing network model for all three examples.

2.2.2.1 Example 1: No Hot-Spots

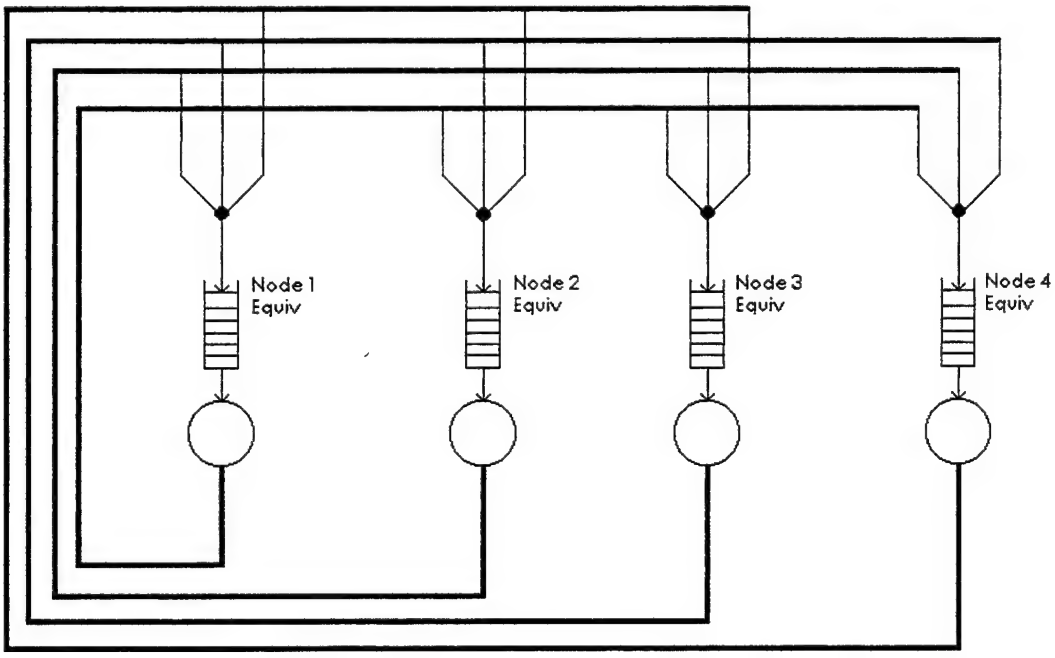


Figure 2.8 Four-Node Equivalent SOME-Bus Model

Since the nodes of Figure 2.7 have the same model as the system presented in Figure 2.1, the load-dependent service rate for the Norton's equivalent of node *i* is already known (Equation 2.12). Substituting copies of this existing node-equivalent service center into Figure 2.7 produces the simplified model of Figure 2.8.

With no hot-spots in the system, the probability transition matrix for the model of Figure 2.8 is shown in Equation 2.25.

$$\mathbf{X} = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 1/3 & 0 & 1/3 \\ 1/3 & 1/3 & 1/3 & 0 \end{bmatrix} \tag{2.25}$$

Solving $\lambda = \lambda\mathbf{X}$ finds all arrival rates to be the same; using Node 4 as the reference, and its arrival rate (μ_{p_r}) as the system parameter, relative arrival rates are established as $\lambda_i = \mu_{p_r}$ $(i = 1,2,3,4)$. For the first two-center aggregation, any two of the identical node-equivalent centers can be chosen to serve as Centers *a* and *b* of Figure 2.5. For reference purposes, the transition probability matrix for this network is seen in Equation 2.26.

$$\mathbf{X} = \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix} \tag{2.26}$$

With arrival rates for nodes a and b already known to equal μ_{p_r}, and their normalization constants equal to those of the reference node (C_a=C_b=C_r), the normalization constants for the aggregate center can be found by applying Equation 2.17.

$$C_{\sigma_{2,1}}(k) = \sum_{j=0}^{k} C_r(j) \cdot C_r(k-j) \qquad (k = 1,2,...,K) \tag{2.27}$$

Using Equation 2.19, the relative arrival rate to the aggregate center is found to be $\lambda_{\sigma_{2,1}} = 4\mu_{p_r}/3$. This results in the load-dependent service rate for the aggregate center shown in Equation 2.28.

$$\mu_{\sigma_{2,1}}(k) = \left(4\mu_{p_r}/3\right) \cdot \left(C_{\sigma_{2,1}}(k-1)/C_{\sigma_{2,1}}(k)\right) \qquad (k = 1,2,...,K) \tag{2.28}$$

Equations 2.20 and 2.21 are used to calculate elements of the reduced system transition probability matrix, as seen in Equation 2.29.

$$\mathbf{X} = \begin{bmatrix} 0 & 1/3 & 2/3 \\ 1/3 & 0 & 2/3 \\ 1/2 & 1/2 & 0 \end{bmatrix} \tag{2.29}$$

The final aggregation, to produce the N-1 node center, substitutes one of the remaining node-equivalent centers for Center *a* of Figure 2.5, and the aggregate center of the previous step for Center *b*. With λ_a equal to μ_{p_r}, and

the relative arrival rate to the aggregate center known to be $\lambda_b = 4\mu_{p_r}/3$ (from the previous aggregation), the normalization constants for this iteration can be found.

$$C_{\sigma_{2.2}}(k) = \sum_{j=0}^{k} C_r(j) \cdot C_{\sigma_{2.1}}(k-j) \qquad (k = 1,2,...,K) \qquad (2.30)$$

Finding the relative arrival rate for this aggregation to be $\lambda_{\sigma_{2.2}} = \mu_{p_r}$, the service rate of the aggregate center is equal to the value shown in Equation 2.31.

$$\mu_{\sigma_{2.2}}(k) = \left(\mu_{p_r}\right) \cdot \left(C_{\sigma_{2.2}}(k-1) \Big/ C_{\sigma_{2.2}}(k) \right) \qquad (k = 1,2,...,K) \qquad (2.31)$$

Equations 2.27 and 2.30 show that normalization constant calculations for aggregate nodes depend only on the normalization constants of individual nodes, or previous aggregations. Also, note that the *N-1* node aggregate service center relative arrival rate, $\lambda_{\sigma_{2.2}}$, equals μ_{p_r}, the arrival rate established for the reference node. This is expected since the final system model of Figure 2.6 places the *N-1* node aggregate in series with the remaining node, which has the same configuration as the reference node. Recognizing these facts provides a computational advantage; there is no need for any calculation during aggregation other than that of aggregate-center normalization constants (Equation 2.17). When aggregation is complete, set the arrival rate of the *N-1* node aggregate equal to the arrival rate of the remaining node.

The product form equation for the final network, which has the same configuration as the network of Figure 2.6, is

$$P(k_p,k_c,k_{agg}) = \left(1/C_{Net}(K)\right)\cdot(1)^{k_p}\cdot\left(\mu_{p_r}/\mu_c\right)^{k_c}\cdot\left(\left(\mu_{p_r}\right)^{k_{agg}}\Big/\prod_{j=1}^{k_{agg}}\mu_{\sigma_{2,2}}(j)\right) \quad (2.32)$$

$$(\text{where } k_p + k_c + k_{agg} = K)$$

producing the network normalization constants shown in Equation 2.33.

$$C_{Net}(k) = \sum_{i=0}^{k}\left[\left[\sum_{j=o}^{i}\left(\mu_{p_r}/\mu_c\right)^{j}\right]\cdot\left(C_{\sigma_{2,2}}(k-i)\right)\right] \quad (k = 0,1,\ldots,K) \quad (2.33)$$

2.2.2.2 Example 2: Hot-Spot Node Performance

Given the same architecture as the first example, the model again simplifies to that of Figure 2.8. The hot spot node for this example is Node 1, as seen in the transition probability matrix of Equation 2.33.

$$\mathbf{X} = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/2 & 0 & 1/4 & 1/4 \\ 1/2 & 1/4 & 0 & 1/4 \\ 1/2 & 1/4 & 1/4 & 0 \end{bmatrix} \quad (2.33)$$

With the hot-spot node the subject of the performance evaluation, the aggregate service center will be composed of all the other nodes. Configuring

the system so that Node 4 is the reference node, solving $\lambda = \lambda \mathbf{X}$, and setting $\lambda_4 = \mu_{p_r}$ results in the relative arrival rate vector $\lambda = [3\mu_{p_r}/2, \mu_{p_r}, \mu_{p_r}, \mu_{p_r}]$. The normalization constants for the first stage of the aggregation are found to be the same as those shown in Equation 2.27, but the aggregate center relative arrival rate is $\lambda_{\sigma_{2,1}} = 3\mu_{p_r}/2$. The normalization constants for the second aggregation are also the same as those of the first example (Equation 2.30). However, the relative arrival rate of that aggregation, the *N-1* node aggregate center, is $\lambda_{\sigma_{2,2}} = 3\mu_{p_r}/2$. Again, note that this arrival rate is equal to that of the remaining (hot-spot) node.

The product-form equation for the final network, represented by the model of Figure 2.6, is

$$P(k_p, k_c, k_{agg}) = \left(1/C_{Net}(K)\right) \cdot \left(3/2\right)^{k_p} \cdot \left(3\mu_{p_r}/2\mu_c\right)^{k_c} \cdot \left(\left(3\mu_{p_r}/2\right)^{k_{agg}} \Big/ \prod_{j=1}^{k_{agg}} \mu_{\sigma_{2,2}}(j) \right) \qquad (2.34)$$

$$(\text{where } k_p + k_c + k_{agg} = K)$$

which results in the network normalization constants of Equation 2.35.

$$C_{Net}(k) = \sum_{i=0}^{k} \left[\left[\left(3/2\right)^i \sum_{j=o}^{i} \left(\mu_{p_r}/\mu_c\right)^j \right] \cdot \left(C_{\sigma_{2,2}}(k-i)\right) \right] \quad (k = 0,1,\ldots,K) \qquad (2.35)$$

2.2.2.3 Example 3: Evaluating a Non-Hot-Spot Node in a Hot-Spot System

For this example, the simplified system model (Figure 2.8), transition probability matrix (Equation 2.33), and relative arrival rate vector are the same as the previous example. For the first aggregation, the hot-spot node is arbitrarily paired with a non-hot-spot node. Substituting the non-hot-spot node for Center *a* and the hot-spot node for Center *b* in Figure 2.5, produces the aggregate center normalization constants shown in Equation 2.36. The term $C_h(k)$ represents the hot-spot node normalization constants.

$$C_{\sigma_{2,1}}(k) = \sum_{j=0}^{k} C_r(j) \cdot C_h(k-j) \tag{2.36}$$

This aggregate center has a relative arrival rate of $\lambda_{\sigma_{2,1}} = 3\mu_p/2$. After reducing the system transition probability matrix, the second aggregation produces the normalization constants

$$C_{\sigma_{2,2}}(k) = \sum_{j=0}^{k} C_r(j) \cdot C_{\sigma_{2,1}}(k-j) \tag{2.37}$$

and the associated relative arrival rate, $\lambda_{\sigma_{2,2}} = \mu_p$, (as expected).

The product-form equation for the final network is

$$P(k_p,k_c,k_{agg}) = \left(1/C_{Net}(K)\right)\cdot(1)^{k_p}\cdot\left(\mu_{p_r}/\mu_c\right)^{k_c}\cdot\left(\left(\mu_{p_r}\right)^{k_{agg}}\Big/\prod_{j=1}^{k_{agg}}\mu_{\sigma_{2,2}}(j)\right) \quad (2.38)$$

$$(\text{where } k_p + k_c + k_{agg} = K)$$

resulting in the network normalization constants shown in Equation 2.39.

$$C_{Net}(k) = \sum_{i=0}^{k}\left[\left[\sum_{j=o}^{i}\left(\mu_{p_r}/\mu_c\right)^{j}\right]\cdot\left(C_{\sigma_{2,2}}(k-i)\right)\right] \quad (k = 0,1,\ldots,K) \quad (2.39)$$

2.2.3 Performance Calculations

When network normalization constants, service rates, and arrival rates are known for the service centers in the model shown in Figure 2.6, performance measures can be calculated. Throughput, already defined in Equation 2.9 for the special case of a single node, is redefined in Equation 2.40 for the network of Figure 2.6. With identical arrival rates to each center (represented by λ_{Net}), individual center throughputs are equal in this serial network.

$$Y_i(K) = \lambda_{Net}\left(C_{Net}(K-1)/C_{Net}(K)\right) \quad \text{where } i = p,c,agg \quad (2.40)$$

The probability of finding *k* messages at a center, given *K* in the network, is defined in Equation 2.41 [8].

$$P(k_i = k) = \left(1 / C_{Net}(K)\right) \cdot \left[C_{Net}(K-k) - \left(\frac{\lambda_i}{\mu_i}\right) \cdot C_{Net}(K-k-1) \right] \cdot \left(\frac{\lambda_i}{\mu_i}\right)^k \quad \text{where } i = p,c,agg \tag{2.41}$$

The utilization of center *i* is then calculated using Equation 2.42.

$$U_i = 1 - P(k_i = 0) \quad \text{where } i = p,c,agg \tag{2.42}$$

The average number of messages at center *i*, given *K* in the network is

$$\bar{k}_i(K) = \sum_{k=1}^{K} k \cdot P(k_i = k) \quad \text{where } i = p,c,agg \tag{2.43}$$

which transforms into Equation 2.44 (by substitution of Equation 2.41).

$$\bar{k}_i(K) = \sum_{k=1}^{K} \left(\frac{\lambda_i}{\mu_i}\right) \cdot \left(C_{Net}(K-k) / C_{Net}(K)\right) \quad \text{where } i = p,c,agg \tag{2.44}$$

The final performance measure, average time spent in the center (Equation 2.45), is determined using a relationship established by Little's Law [36] as practiced in a queueing network analysis method known as Mean Value Analysis [47].

$$\bar{t}_i(K) = \bar{k}_i(K) / Y_i(K) \quad \text{where } i = p,c,agg \tag{2.45}$$

These formulas are applied in the next chapter to analyze the performance of the SOME-Bus message-passing model as the number of nodes, service rates, and number of messages are varied.

Chapter 3

MESSAGE-PASSING MODEL PERFORMANCE EVALUATION

This chapter presents the results of analytical and simulation modeling of a message-passing SOME-Bus system along with performance comparisons to crossbar and torus simulations. Performance measurements provided by the queueing-network model developed in Chapter 2 are used to evaluate processor utilization and communication latency, as well as validate a SOME-Bus simulator with the same average service times and distributions. The simulator is then extended to perform commonly occurring system activities that are difficult or impossible to model analytically. Results from the modified SOME-Bus simulator are compared to simulations of crossbar and torus systems to determine the relative performance of the SOME-Bus architecture with respect to these two network topologies.

3.1 Analytical Model Results

The analytical model evaluated here is the *N*-node SOME-Bus queueing-network model that was developed in Chapter 2 (shown in Figure 2.1). The model consists of a network of service centers that represents system resources, with each node modeled as two service centers in tandem. The first service center represents the activities of the receiver-buffer and processor-memory blocks of the SOME-Bus architecture, as depicted in Figure 1.1. The second service center models the actions of the transmitter block and the channel

characteristics of the same figure. All service centers in the model are assumed to have load-independent, exponential service time distributions, and first-come, first-served service disciplines.

The parameters of the model that can be manipulated include the number of nodes (*N*), the average processing service time, average channel service time (which is proportional to message size), the total number of processes executing in the system $(N \cdot K)$, and the destination node selection distribution. The primary means of comparing the performance of the analytical and simulation models involves examination of processor utilization (average fraction of time dedicated to executing processes) and communication latency (channel queueing plus transfer time).

The reference point for all timing parameters and measurements is the average time assigned to service a process (t_p), maintained at 100 time units for all model runs. The average channel service time (t_c) represents the message transfer time through the channel and is varied between 5 and 100 time units. All performance data is presented so that the x-axis of the associated plot represents the ratio of average message transfer time to average process service time (t_c/t_p), a ratio indicative of the amount of communication overhead incurred per unit of computation [52]. When the value of t_c/t_p is low, the task granularity is assumed to be coarse, while higher values indicate more finely-grained tasks.

Figure 3.1 shows analytical model (T) processor utilization for a SOME-Bus system as the number of nodes in the model is varied from 4 (-4N) to 128 (-128N) (in powers of 2), with the number of processes averaging three-per-node. Note

that the difference in processor utilization, as communication service time is varied, remains at less than 5 percent between a 4-node and a 128-node configuration. In addition, the difference in utilization as the number of nodes increases converges on the 128 node curve, with the difference between any two of the curves remaining constant over the entire range of t_c/t_p. Taken together, these facts indicate that the system scales well, with even the largest system operating at the finest granularity, being able to maintain processor utilization above 60%.

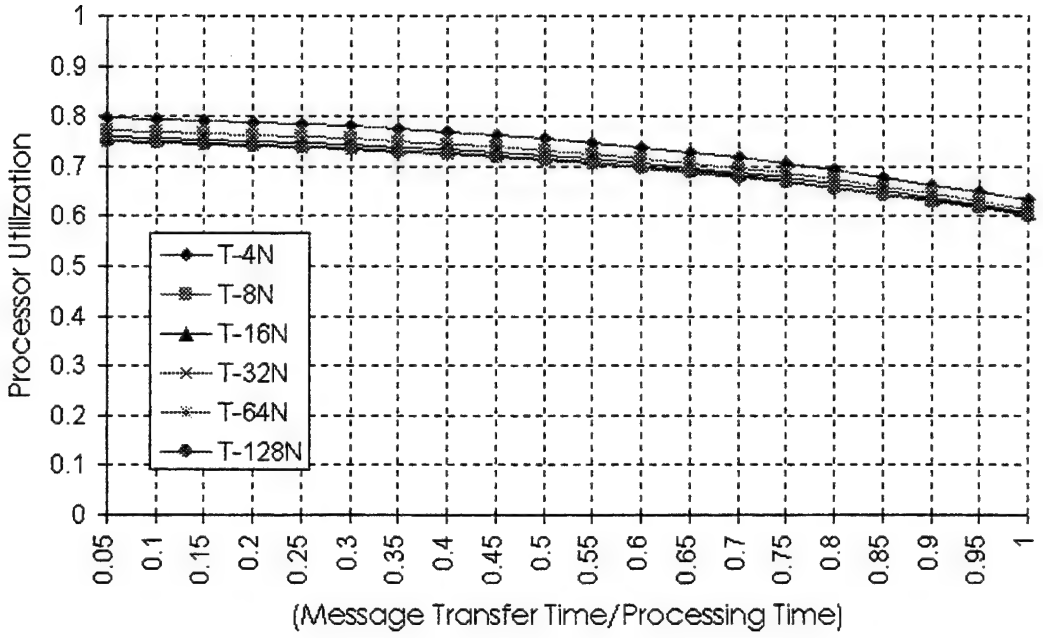


Figure 3.1 SOME-Bus Processor Utilization, 3 Tasks-per-Node, Analytical

Figure 3.2 presents an expanded view of 4-, 16-, and 64-node SOME-Bus processor utilization curves, showing both analytical (T) and simulation (S) results.

The close correspondence between the two sets of curves indicates the simulator provides accurate results for processor utilization as the number of system nodes is varied. There is also similar correspondence between analytical and simulation results for 8-, 32-, and 128-node system configurations.

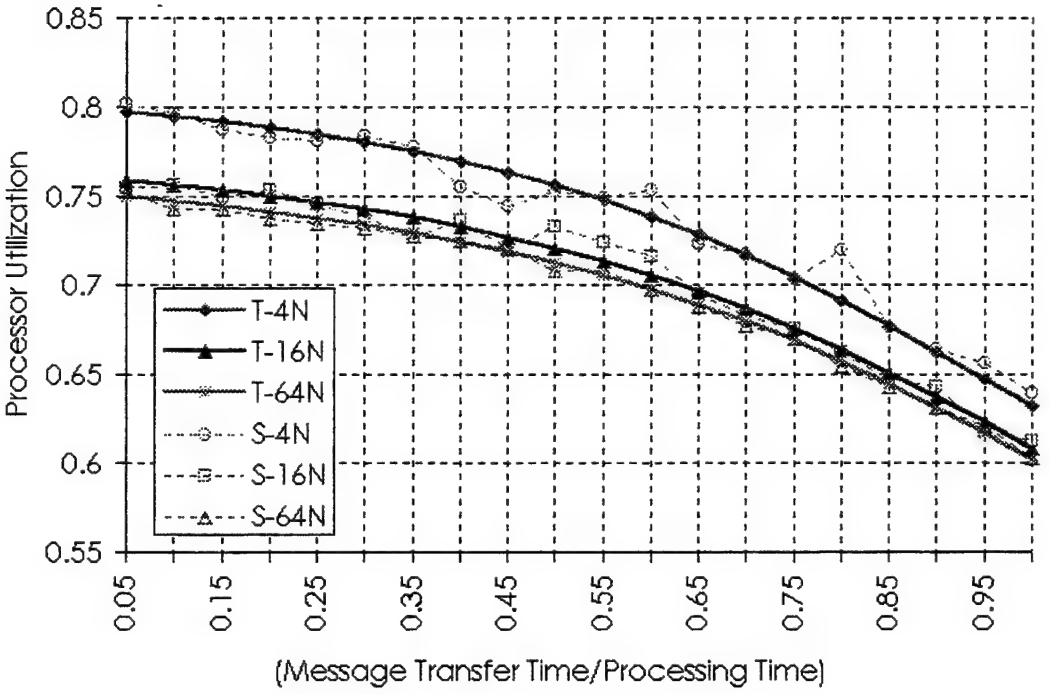


Figure 3.2 Processor Utilization, 3 Tasks-per-Node, Analytical and Simulation

Further proof of simulator accuracy is provided in Figure 3.3, where analytical (T) and simulation (S) results are shown as the number of nodes in the SOME-Bus is fixed at $N = 64$, and the number of processes is varied between 64 and 320, in units of 64. These numbers correspond to an average of 1 (-1P) to 5 (-5P) processes-per-node. Note that processor utilization does not decline as the number of processes is increased, an indication that the network can effectively

deal with the additional message traffic generated by multiple processes-per-node. Also, as task granularity increases, the difference between curves remains constant, an additional indication that network attributes have minimal impact on the ability of a processor to execute its tasks.

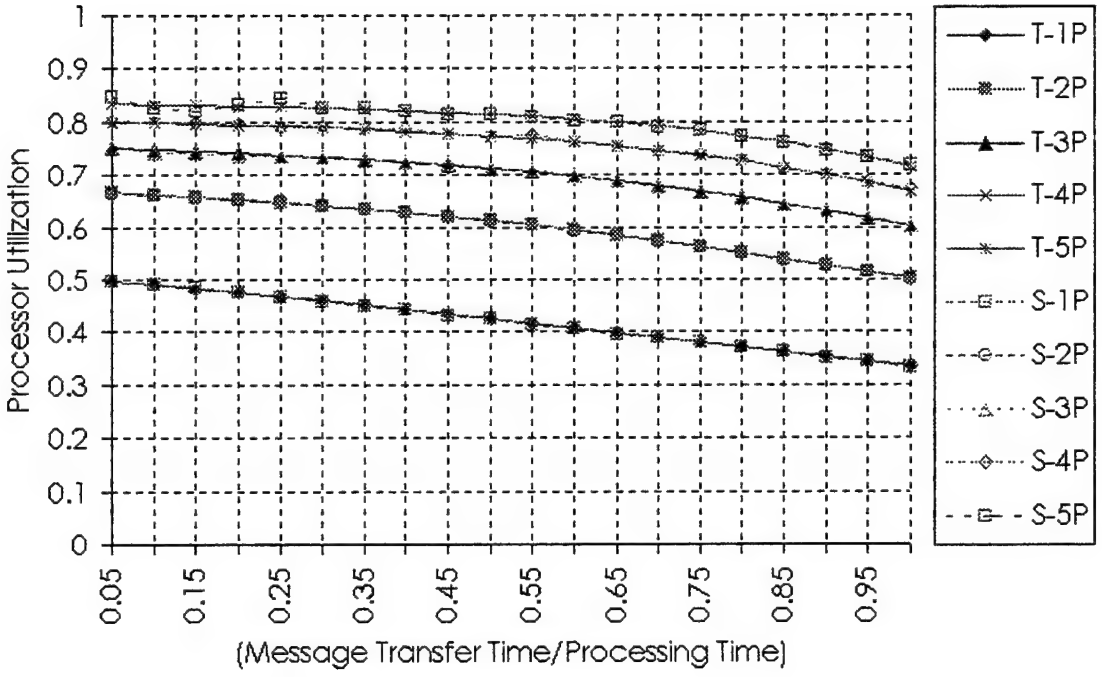


Figure 3.3 64-Node SOME-Bus Processor Utilization,1 to 5 Tasks-per-Node

Processors in message-passing systems depend on their interconnection network to provide, in a timely manner, the data needed to enable their assigned processes. One critical measure of this ability is communication latency. In the SOME-Bus analytical model, this measure consists of the time a message spends in the channel center, both waiting and in service. Channel service time in the analytical model corresponds to message transfer time in the

simulation, due to the constant channel bandwidth of a SOME-Bus channel. Figure 3.4 shows communication latency for SOME-Bus systems as the number of nodes is varied from 4 to 128 nodes (in powers of two), and the number of processes is kept fixed at $K = 192$. In addition, the simulation results for 4-, 16-, and 64-node systems are included for validation purposes. This set of curves shows that latency does not depend on the number of nodes in the system, and that analytical and simulation models are again in close agreement.

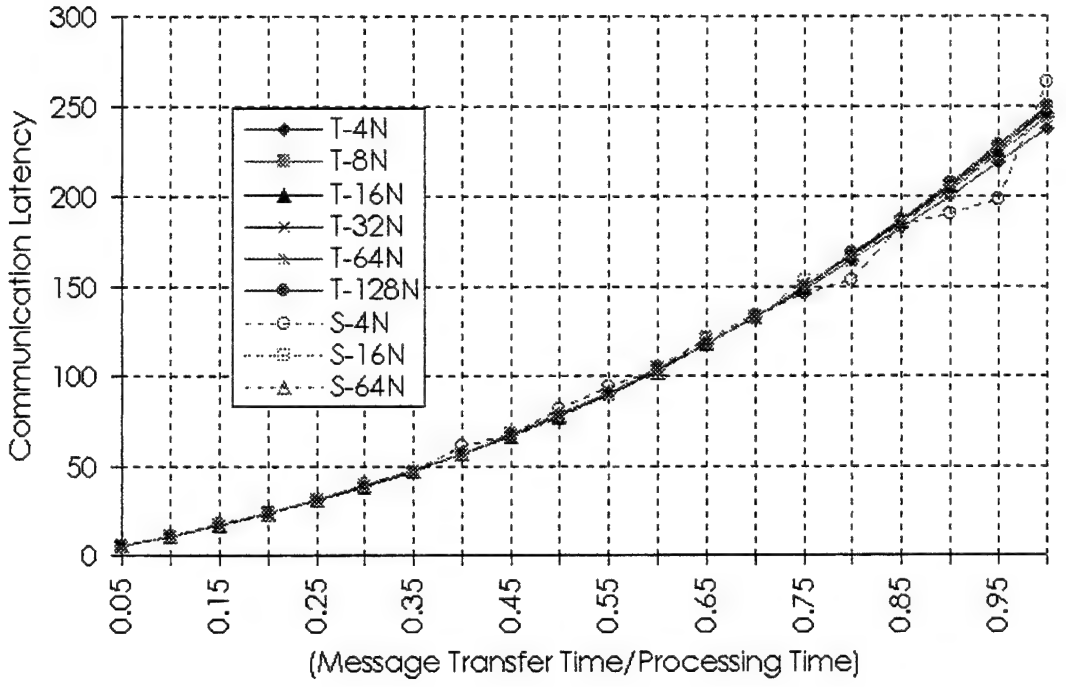


Figure 3.4 Communication Latency, SOME-Bus, 3 Tasks-per-Node

Figure 3.5, shows communication latency for a 64-node SOME-Bus system as the number of processes is varied from 64 (-1P) to 320 (-5P), in units of 64. Both analytical (T) and simulation (S) results appear on the graph, again in close

agreement. When tasks are coarse-grained, there is essentially no waiting time in the channel queue, regardless of the number of processes. As the ratio of communication to computation increases, the processor becomes less of a bottleneck and waiting times in the channel queue add to the increase in latency.

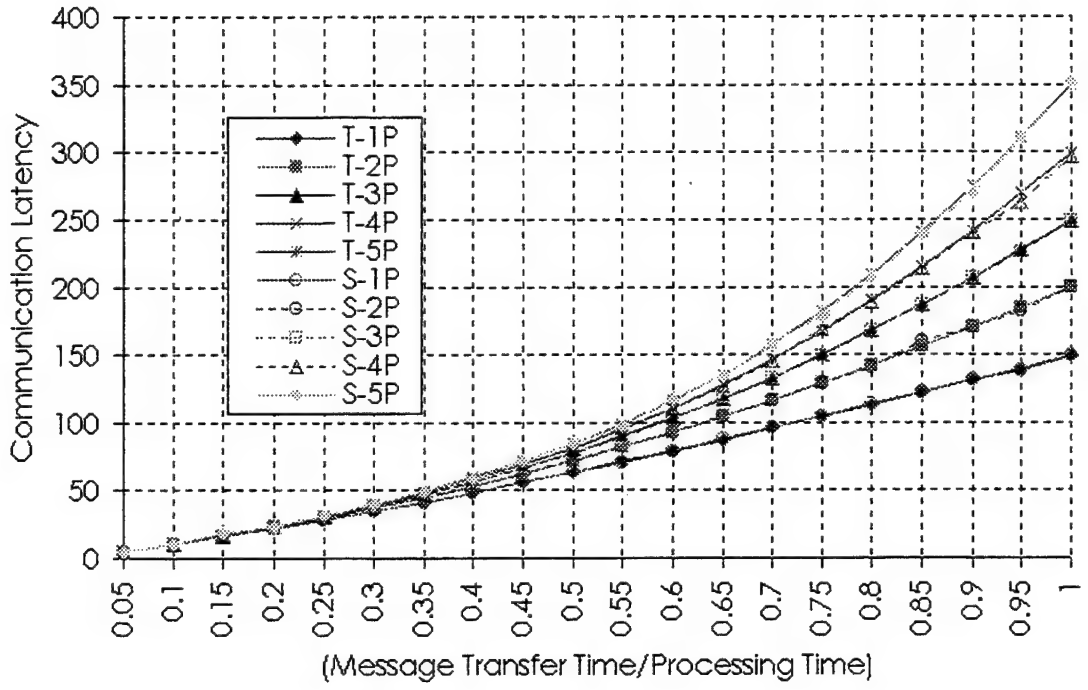


Figure 3.5 Communication Latency, 64-Node SOME-Bus,1 to 5 Tasks-per-Node

3.2 Comparison to Torus and Crossbar Systems

The close correspondence between utilization and communication latency results of the message-passing SOME-Bus analytical and simulation models validates the ability of the SOME-Bus simulator to predict system performance. This section presents results from a modified version of that

simulator, comparing them to simulations of two-dimensional torus and crossbar message-passing systems. All modifications represent commonly occurring processing or communication activities that are difficult or impossible to model analytically. One of these modifications permits examination of system performance when task synchronization is required.

Task synchronization is modeled in all three types of networks by having each node send the same synchronization message to all other nodes in the form of an all-to-all communication. Processors initiate a synchronization operation after a specified period of time elapses and the processor completes its current processing task. Synchronization intervals considered here are 5000 time units (representing heavy synchronization, synchronizing an average of once every 50 messages), or 30000 time units (representing light synchronization, averaging once every 300 messages). In addition, each synchronization operation can be preceded by a processor sending a number (D) of distinct data messages to other nodes. Each of these messages is short in duration (5 time units), with D equal to 0 or 10. This activity represents the exchange of partial results that normally occurs prior to task synchronization.

Processing, as defined for these models, consists of the processor at each node extracting a data message from an input queue, servicing the associated process for a period of time, generating an output data message, and then suspending that process. A processor operating in this manner becomes idle only when all processor input queues are empty. Also, in all architectures, the receiver or router at each node is assumed to contain dedicated synchronization hardware. This assumption releases the processor from having to

dedicate part of its service to synchronization tasks, and eliminates synchronization messages from the processor input queue(s).

In the crossbar architecture, each node is able to connect its single output channel to the single input channel of any other node. Messages wait at the output channel queue of a source node if the input channel of the destination is engaged in communication with a different node. When a node needs to broadcast the same message to all nodes (modeled here as a synchronization operation), it waits until all input channels are free, reserves them, and then simultaneously sends a single copy of the message to all nodes.

In the torus architecture, dedicated channels connect each node to its four adjacent neighbors. Each of those four channels has an associated output queue to buffer messages traveling through the network from source to destination. In addition, each node has a single input queue to buffer messages directed to that node. When source and destination nodes are not nearest neighbors, an adaptive form of wormhole routing is employed to facilitate communication. This method routes the message through the next available channel in the direction of the destination. Synchronization is performed by having all nodes send a synchronization message to a barrier node. Each node in a row receives a single synchronization message on its right channel, forwarding this message to its left channel once it arrives at its barrier. A similar procedure forwards the synchronization message to the barrier node through the left column. Once the barrier node accumulates synchronization messages from all nodes, it redistributes them in the opposite fashion to reinitiate processing.

In the SOME-Bus architecture, messages leaving a node are broadcast from the associated channel queue through the corresponding channel in a single operation. The processor operates in a fashion similar to a server in a polling system. Polling system theory indicates that service discipline has an effect on the waiting time of the messages in the input queues. In the simulations presented here it is assumed that the processors employ a limited service discipline. Synchronization messages are processed in the receiver at the destination node, and their arrival is marked in a local table.

With respect to all three architectures, the processing time, t_p, again serves as the reference point for all performance evaluations, being geometrically distributed with a mean of 100 time units. The time it takes to transfer a message, t_c, is varied from 3 to 100 time units, and is uniformly distributed. This transfer time represents actual channel time in crossbar and SOME-Bus simulations, but it only represents actual channel time for the torus when a message is not blocked in transit from source to destination. In addition, establishing individual channel connections for the wormhole routing scheme of the torus requires a small amount of time. Destination nodes in the crossbar and SOME-Bus are uniformly selected over all nodes (excluding the source) and uniform in each direction in the torus. All architectures are assumed to have the same channel capacity, and all simulations were performed with the number of nodes (N) set equal to 64 and the number of processes in the system set equal to 192.

Although all the simulations performed here use the same channel capacity for each of the three architectures, it is not necessarily true that they

have the same transmission-rate capability. To illustrate, if the processor subsystem of a node in the torus architecture utilizes typical high-end microprocessors with moderate-size caches and single-bus access to a moderate-speed main memory, memory bandwidth can be the factor which limits the speed at which messages are transmitted and received. Many high performance all-to-all communication (and synchronization) algorithms assume that nodes in a torus are capable of all-port communication. This assumption limits the transmission rate of torus network channels to one-quarter the memory bandwidth, possibly less if the node is also configured to accept incoming messages at similar rates. The SOME-Bus receiver, with a dedicated buffer for each input channel, and the crossbar, with its single input channel and buffer, avoid this restriction. As an example, given messages of size M bytes and a memory bandwidth of B bytes/sec, it takes $4M/B$ seconds to transmit the message from node-to-node in the torus. Given the same message size in a SOME-Bus or crossbar system, it takes $2M/B$ seconds to transmit (due to the buffer between the channel and memory). These basic calculations show that for equal processing capability, channel capacity, and memory bandwidth, the SOME-Bus and crossbar systems can operate at twice the transmission rate of torus channels.

Figure 3.6 shows processor utilization for the three architectures (SO = SOME-Bus, CR = crossbar, TO = torus) as message transfer time is varied and no synchronization operations occur. Results indicate that, in the absence of synchronization, all architectures display similar performance for short to medium message transfer times. SOME-Bus and crossbar system performance appear to

be identical, showing only minor degradation as message transfer times increase. As the time it takes to transfer a message approaches the computation time of the associated process, processor utilization in the torus decreases at a markedly greater rate than the other two architectures. These results are in general agreement with studies of similar architectures in [1].

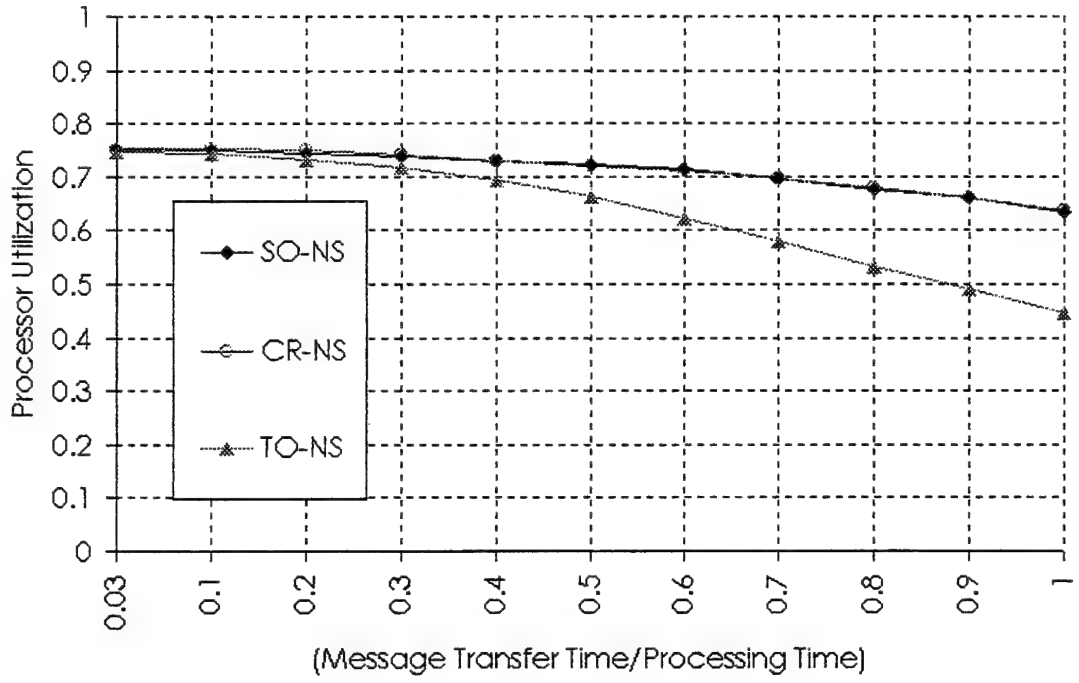


Figure 3.6 Processor Utilization, All Architectures, No Synchronization

Figure 3.7 illustrates processor utilization for all three architectures in the presence of synchronization operations. Cases of both heavy synchronization (HS), where synchronization occurs every 5000 time units (averaging 50 messages between synchronizations), and light synchronization (LS), where synchronization takes place every 30000 time units (averaging 300 messages between

synchronizations), are shown. In addition, separate simulations were performed with the number of short data messages, representing an exchange of intermediate results just prior to transmission of synchronization messages, set at 0 (0) and 10 (10). The SOME-Bus outperforms the other two architectures in all four situations, seemingly unaffected by the presence of an exchange of intermediate results prior to synchronizations. The SOME-Bus also maintains a constant difference between HS and LS curves. In contrast, heavy synchronization operations, coupled with longer message transfer times, appear to have a dramatic effect on the ability of torus and crossbar systems to keep their processors occupied with assigned tasks.

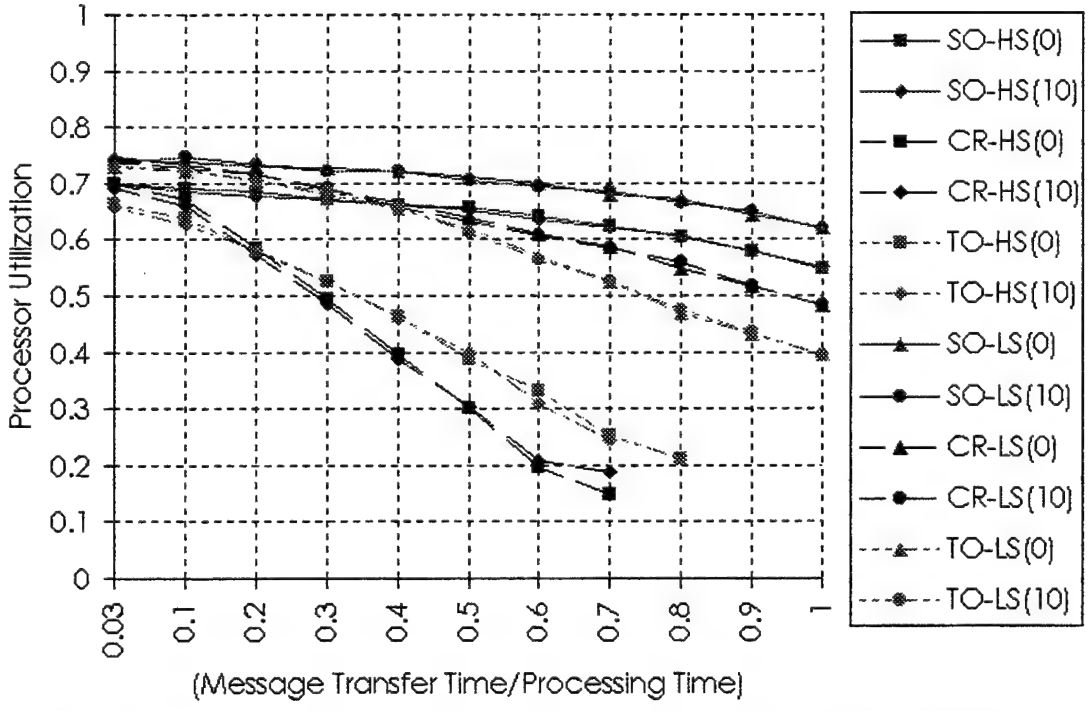


Figure 3.7 Processor Utilization, All Architectures, with Synchronization

Figure 3.8 shows communication latency (time a message is queued and in transit as it travels from source to destination) when no synchronization operations occur, and average message transfer times are varied. Again, the SOME-Bus and crossbar systems exhibit identical performance, while the torus system experiences significant additional delays as messages increase in size.

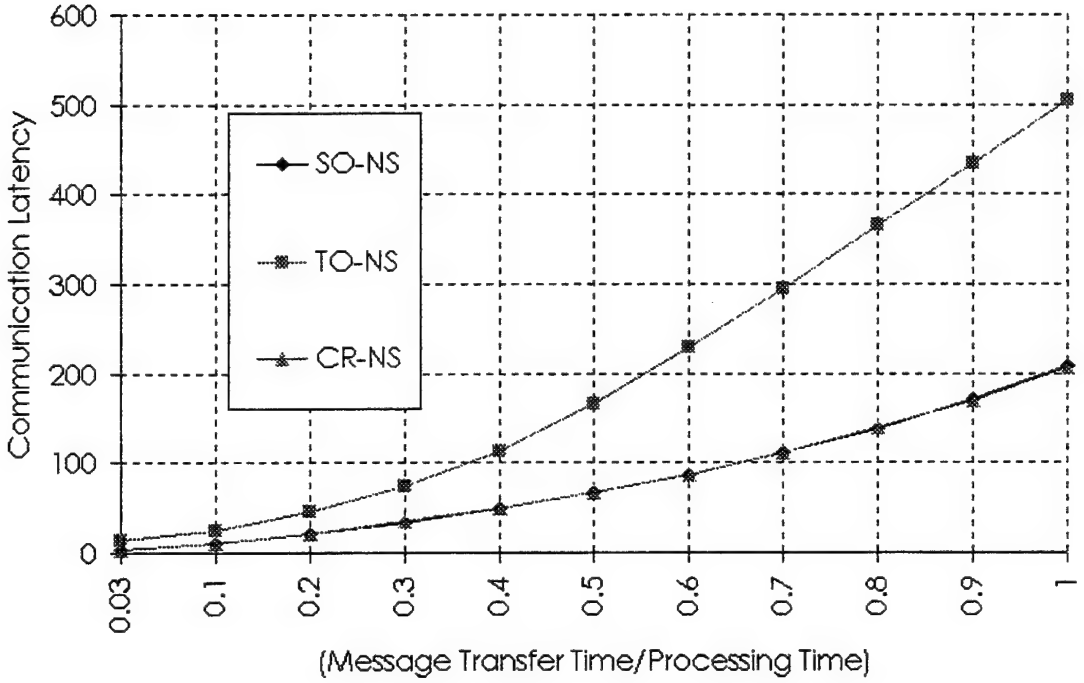


Figure 3.8 Communication Latency, All Architectures, No Synchronization

Communication latency for the three architectures in the presence of synchronization is shown in Figure 3.9. Whether heavy or light synchronization occurs, and whether or not intermediate results are exchanged, the SOME-Bus exhibits the same performance as when no synchronization operations occur

(Figure 3.8). In comparison, the crossbar is significantly effected by heavy synchronization operations, and even light synchronization noticeably increases latency with respect to no synchronization. The torus has at least twice the latency, on average, as the SOME-Bus, regardless of the type of synchronization that occurs. The SOME-Bus clearly has less latency, in all synchronization situations, than the crossbar or the torus.

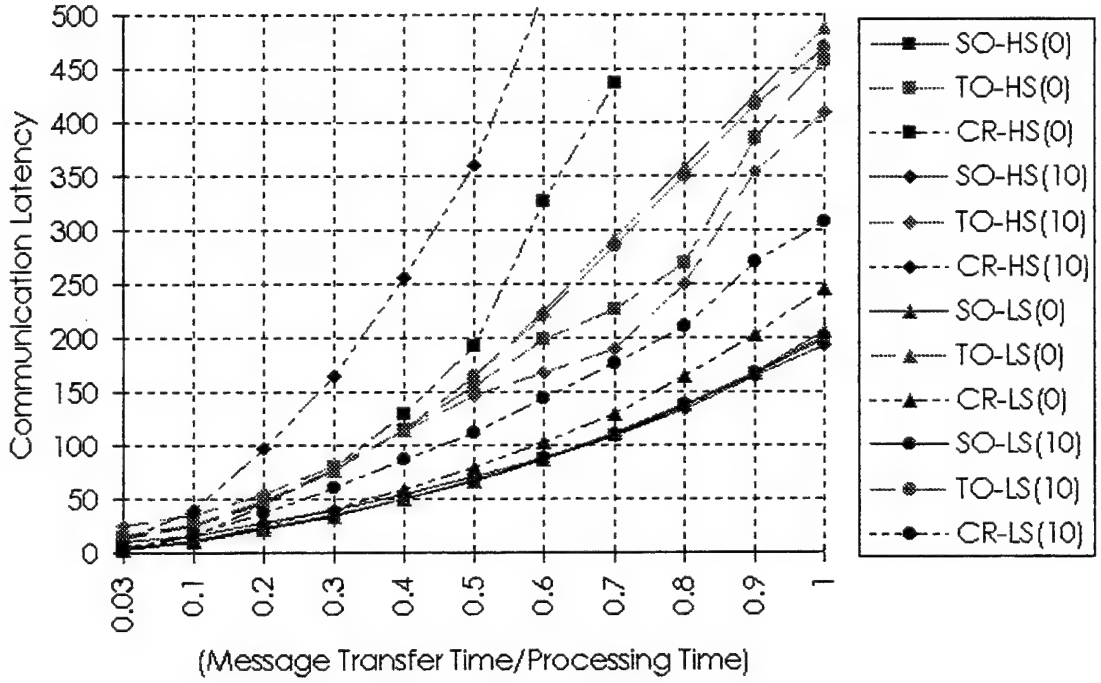


Figure 3.9 Communication Latency, All Architectures, With Synchroniztion

As mentioned earlier in this section, these simulations do not take into account the fact that SOME-Bus and crossbar channels should have twice the transmission-rate capability of torus channels. Taking this fact into account, the SOME-Bus compares even more favorably with the other architectures.

Chapter 4

DISTRIBUTED SHARED MEMORY MODELS OF THE SOME-BUS

4.1 Background

As discussed in Chapter 1, contemporary shared-memory multiprocessor systems are designed to benefit from the technological advances in, and economic advantages of, commercial microprocessors and their support components [50]. The majority of those microprocessors have on-chip cache memory to improve uniprocessor performance by bridging the speed gap between processor and memory technology. When processors take advantage of this feature in multiprocessor systems, cache coherence becomes a major concern.

Snooping is a common hardware-based technique to maintain coherence in central-shared-memory systems, informing all caches of every memory write by every processor. Existing distributed-shared-memory (DSM) systems are unable to implement snooping because their interconnection networks quickly saturate due to the additional message traffic associated with the method [44]. Software-based cache-coherence techniques are available for DSM systems, but also incur excessive communication overhead, limiting them to systems with few processors. Another method to implement coherence in DSM systems is a directory-based protocol, where the directory catalogs the state of every block that is cached. The overhead associated with this method scales acceptably in systems with up to one hundred processors [44].

Because of the difficulties involved in maintaining cache coherence, some DSM multiprocessors, such as the Cray T3D, make shared data uncacheable, using the cache only for private data [44]. These DSM systems fall into the non-uniform memory access (NUMA) category of architectural models. The SOME-Bus, when functioning as a DSM multiprocessor, can readily support cache coherence, placing it in the cache-coherent, non-uniform memory access (CC-NUMA) category. Although the SOME-bus can utilize software or directory-based techniques for implementing cache coherence, its design allows the transmitter, receiver, and cache controller to function as a hardware-based, integrated cache-coherence mechanism.

The SOME-Bus cache-coherence mechanism functions by having every node broadcast messages to update, or invalidate, remote caches when a processor writes to local memory. Receivers at remote nodes monitor these messages, signaling their cache controller when one is detected. This hardware-based approach enforces coherence at the cache-block level, reducing the probability of false sharing and thrashing.

Although the non-blocking nature of the SOME-Bus network eliminates the possibility of interconnection network saturation, intense cache-consistency traffic could saturate the cache controller [24]. In that situation, the SOME-Bus cache-coherence mechanism could leverage existing directory-based techniques, only notifying remote caches with affected data blocks. This would be accomplished by including a list of destinations in the invalidation-message header, making the decision to accept or reject an input message the responsibility of the receiver, rather than the cache controller. This same

mechanism would allow a SOME-Bus-based system to support multiple-read, multiple-write data accesses. A write by a cache controller to the shared address space in local memory could automatically be propagated to all other nodes, updating cached data in a non-demand, anticipatory manner.

4.2 DSM Model Development

As the number of nodes in a distributed-shared-memory system grows, a greater percentage of the shared address space resides outside the local node. In this situation it is reasonable to assume that the miss rate to local memory will increase, with support for this assumption found in other multiprocessor-system studies [56], [12]. Given finite memory bandwidth, this increased rate of remote-memory requests can be expected to interfere with the ability of memory to service local requests. The mechanisms that influence this interference, which reduces system performance, are the focus of the SOME-Bus DSM models.

Two queueing-network models will be developed to examine performance of the SOME-Bus as a DSM system. These models, again based on the system architecture shown in Figure 1.1, represent an *N*-node SOME-Bus, with two service centers used to model the subsystems of a node. The first service center, referred to as the processor service center, represents the combined activities of the receiver, processor (with cache), and memory subsystem. The second service center, referred to as the channel service center, represents the behavior of the transmitter and output channel. In addition, multithreaded execution is assumed, with the processor at node *i* responsible for executing *K* (constant) threads. When performing tasks associated with the execution of its

own threads, a node is referred to as a local node; when performing tasks associated with the threads owned by another node, it is referred to as a remote node.

In the actual system, activity at a local node begins with the execution of a thread, and continues until a cache miss occurs. If the miss can be supported by local memory, the thread continues to execute. If remote data access is required the thread suspends, a read request is generated, and that request is transmitted to the remote node hosting the memory address. Simultaneously, a context switch occurs if any other threads are ready to execute.

To model local node activity, consider the case where k_i, the number of outstanding remote memory requests by node i, is less than K, the number of processing threads assigned to a node. In this situation it is assumed the processor continues to generate remote-memory requests, with a mean interval of h time units between requests. This interval corresponds to the processor-center service time (exponentially distributed with mean h) that each of the remaining K- k_i threads receives. These assumptions are similar to those made in other performance studies involving DSM systems based on mesh [1], torus [42], and multistage interconnection networks [56]. Once a request message is generated, it is placed in the queue of the channel service center for transmission on the output channel. When the message reaches the server of the channel service center, it requires a mean of s time units (exponentially distributed) for transmission, where s is proportional to the length of the message. (This assumption is based on the fact that a SOME-Bus has constant channel

bandwidth.) It is further assumed that remote-memory requests are directed with equal probability to the other nodes in the system.

Modeling of remote-request activities begins with the message in the processor-center queue at a remote node. Once the message reaches the head of the queue, it requires a mean service of *m* time units (exponentially distributed) to access the data and assemble the response message. Again, the output of the processor service center is placed in the channel service center queue, and once it reaches the server, a mean of *s* time units is required for transmission. Remote-memory-request responses are returned to their owner node.

Referring again to the actual system, when a remote-memory request reaches a node, a DMA controller in the memory subsystem performs the necessary memory accesses, creates the response message, and forwards it to the transmitter. While performing these tasks, the DMA controller competes with the cache controller of the same node for local memory bandwidth. Since this competition can be managed in different ways, there is a need for two DSM models.

The first model contains a single queue for both remote-memory request and response messages. Arriving messages are serviced in a first-come, first-served manner, which corresponds to the processor in the actual system having exclusive access to local memory while a thread executes. As a result, remote-memory requests are serviced only at context switch points when they reach the head of the queue. The second model contains separate queues for remote-memory requests and responses, with the message at the head of each queue

receiving a fixed portion of the available service. This model, in effect, represents sharing of local memory bandwidth as remote-memory and thread-processing requests interleave their local memory accesses.

Upon initial inspection, it appears that the two DSM SOME-Bus queueing-network models will each require *N* nodes, and that the number of customer classes should be a function of both the types of messages and the number of nodes, with each class having population *K*. However, since no distinction is made regarding the origin of remote-memory request messages and responses are returned directly to the node making the request, the models can be abstracted into two-node, two-class (remote-memory request and response) aggregates.

4.2.1 Model 1

In the first model of a SOME-Bus-based DSM multiprocessor, illustrated in Figure 4.1, both message classes wait in the same input queue at the service centers and are served in a first-come, first-served (FCFS) manner. In the case of the processor service center, this corresponds to the situation where all memory bandwidth is allocated to local threads as they execute, and remote requests for data are serviced only when they reach the head of the queue. Because the service time for thread processing at the processor service center, *h*, can differ from the service time to generate a response message to a remote-memory request, *m*, a product-form solution does not exist for this network [3]. To calculate performance measures for this model an approximation technique, based on an evaluation method known as mean value analysis (MVA), is used

[46]. Mean value analysis provides a method to recursively compute the mean number of customers (messages), center residence times (queued plus service), and throughputs for individual service centers in a closed queueing network. The approximation method developed in [46], and used here, generates the same values, but does so by iterating until convergence is achieved.

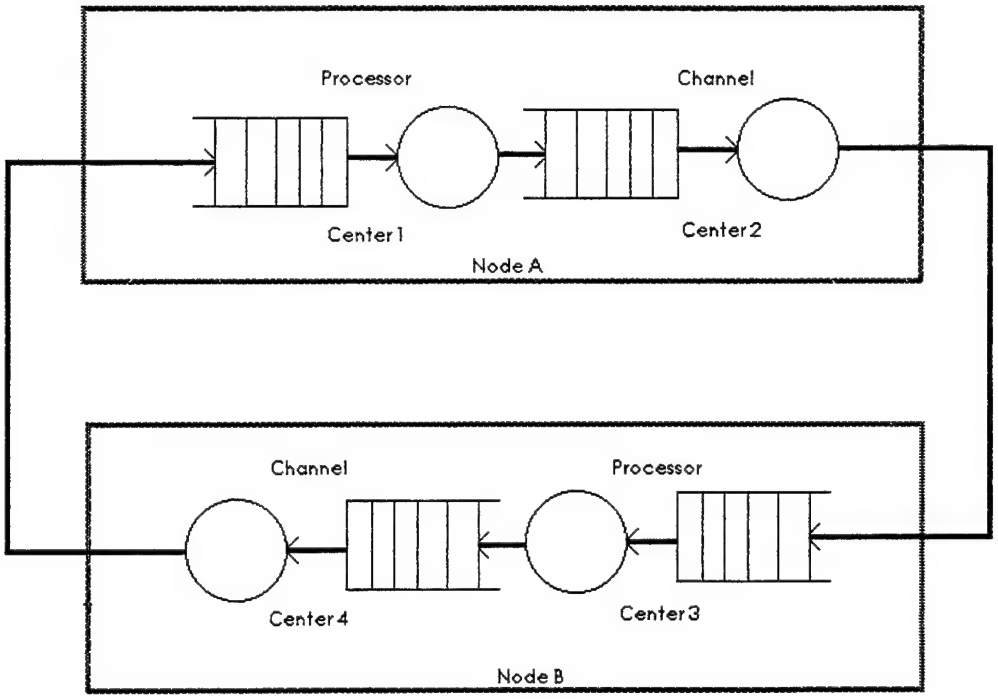


Figure 4.1 SOME-Bus DSM Multiprocessor Model 1

There are two chains present in the model of Figure 4.1: the first (Chain 0) consists of request messages issued by Node A and responses by Node B, while the second (Chain 1) consists of request messages issued by Node B and responses by Node A. Let τ_{i_j} represent the service time that service center *i*

applies to messages of chain *j*. The service times associated with the four service centers of Model 1 are shown in Equation 4.1.

$$\begin{aligned} \tau_{1_0} &= \tau_{3_1} = h \\ \tau_{1_1} &= \tau_{3_0} = m \\ \tau_{2_0} &= \tau_{4_0} = \tau_{2_1} = \tau_{4_1} = s \end{aligned} \tag{4.1}$$

The arrival theorem [31] that serves as the basis for MVA [47] states that a message of chain *r*, arriving at service center *i*, finds an equilibrium situation equivalent to that of a network with one less customer (message) in the arriving message's chain. This has been proven true when the network has a product-form equation. The MVA approximation technique developed by Reiser [46] is based on the postulation that this theorem holds true even for non-product-form networks.

Let $n_{i_j}(\mathbf{r}-)$ represent the mean number of chain *j* messages in service center *i* just prior to the arrival of a message from chain *r*. Then

$$t_{i_r} = \tau_{i_r} + \sum_{j=0}^{1} \tau_{i_j} \cdot n_{i_j}(\mathbf{r}-) \tag{4.2}$$

is the mean time a message from chain *r* spends at service center *i* (queued and in service) when the service centers have exponential service times [46]. If the system has a product-form solution, the exact value of $n_{i_j}(\mathbf{r}-)$ is found by removing one message from chain r, as seen in Equation 4.3. In this equation **K**

represents the chain population vector ($\mathbf{K} = (K_0,K_1)$ for the model of Figure 4.1) and $\mathbf{e}_r$, a unit vector with the 1 in the r^{th} position.

$$n_{i_j}(\mathbf{r}-) = n_{i_j}(\mathbf{K} - \mathbf{e_r}) \tag{4.3}$$

When only an approximate solution is possible, $n_{i_j}(\mathbf{r}-)$ must be estimated. Let $\varepsilon_{i_j}(\mathbf{r}-)$ represent the difference in the average number of messages in chain *j* at station *i*, as seen in Equation 4.4.

$$\varepsilon_{i_j}(\mathbf{r}-) = n_{i_j}(\mathbf{K}) - n_{i_j}(\mathbf{K} - \mathbf{e_r}) \tag{4.4}$$

To determine the value of $\varepsilon_{i_j}(\mathbf{r}-)$ for a specific chain, evaluation of a single-chain MVA is required. First, suitably redefined service rates for each center in that chain must be determined. Equation 4.5 shows that the effective single-chain service rate for a chain *r* message at service center *i* results from the fraction of service center capacity that a chain *r* message uses, the remainder devoted to other chains [46].

$$\hat{\tau}_i = \tau_{i_r} \Big/ \left(1 - \sum_{j \neq r} \lambda_j \tau_{i_j}\right) \tag{4.5}$$

Upon completion of the single-chain MVA, $\varepsilon_{i_j}(K)$ is found using Equation 4.6.

$$\varepsilon_{i_r}(K) = n_{i_r}(K) - n_{i_r}(K-1) \tag{4.6}$$

Assume that only the chain with the arriving message is significantly affected by the $\varepsilon_{i_j}(\mathbf{r}-)$ term in Equation 4.4 (therefore set $\varepsilon_{i_j}(\mathbf{r}-) = 0$ when $j \neq r$), allowing the substitution of Equation 4.6 for Equation 4.4. As a result, Equation 4.2 simplifies to Equation 4.7, establishing the approximate waiting time for a chain *j* message at service center *i* [46].

$$t_{i_r} = \tau_{i_r} + \tau_{i_r} \cdot (n_{i_r}(K) - \varepsilon_{i_r}(K)) + \tau_{i_q} \cdot n_{i_q}(K) \qquad (\text{where } q \neq r) \tag{4.7}$$

Completing the multichain MVA approximation, the throughput of chain *j* is

$$\lambda_j \leftarrow K \Big/ \sum_{i=1}^{4} t_{i_j} \tag{4.8}$$

resulting in the mean number of chain *j* messages at center *i* shown in Equation 4.9.

$$n_{i_j} \leftarrow \lambda_j \cdot t_{i_j} \tag{4.9}$$

Summarizing the approximation technique in algorithmic form [46]:

1. Initialize n_{i_j} and λ_j for each chain. The only requirement for distributing the messages among the centers is that each chain has the correct number of messages. A suggested initial value for λ_j is the lowest service rate of all centers in the chain.
2. Repeat steps 3-6 until convergence occurs.
3. Perform single-chain MVA for both chains to determine $\varepsilon_{i_j}(K)$ (Equations 4.5 and 4.6).
4. Calculate mean waiting times at each center (Equation 4.7).
5. Calculate chain throughputs (Equation 4.8).
6. Calculate mean number of messages at each center (Equation 4.9).

4.2.2 Model 2

The second DSM SOME-Bus model, shown in Figure 4.2, is again abstracted to two-nodes, but with separate queues for the two chains at the processor service centers. The separate queues are used to represent the case where a DMA and cache controller interleave their memory accesses. As in Model 1, Chain 0 receives thread processing at Node A, producing remote-memory requests, and DMA service at Node B, generating response messages. Correspondingly, Chain 1 receives DMA service at Node A and thread processing at Node B. Thread and remote-memory request service times at a processor center can differ, with *h* representing the mean thread processing time, and *m* the mean time to process a remote-memory request (both

exponentially distributed). At the channel service center both chains receive the same exponential service, with a mean of s time units. If messages from only one chain are enqueued at a processor center, the message at the head of the corresponding queue receives service at its full service rate. When messages from both chains are present, the messages at the head of each queue share available service, with the fraction α_T going to the message in the thread queue and α_R to the message in the request queue ($\alpha_T + \alpha_R = 1$).

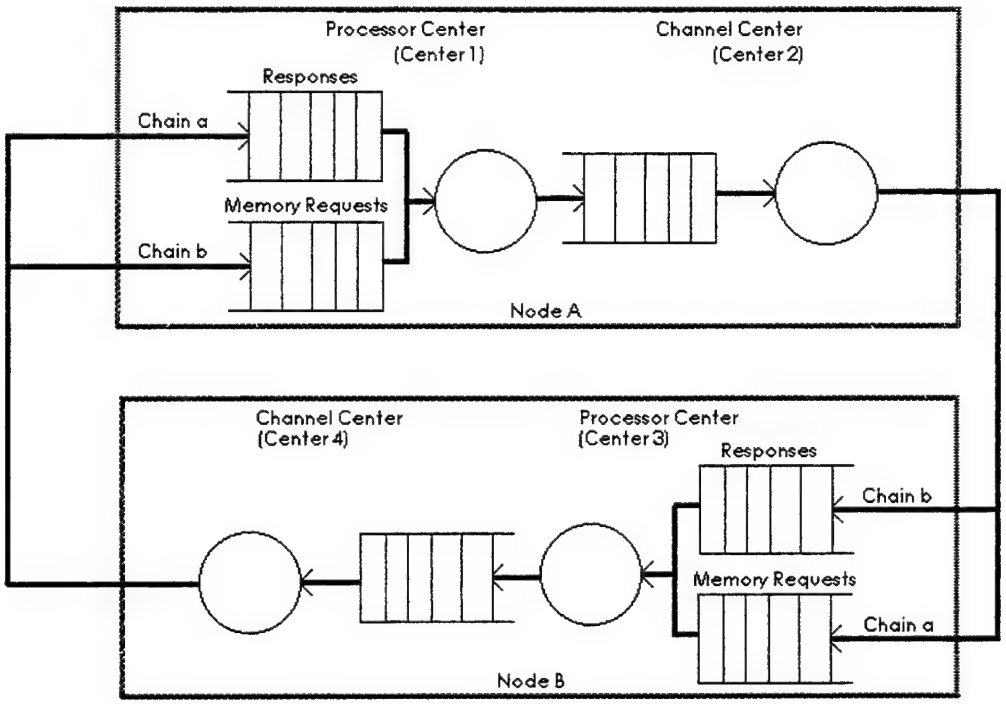


Figure 4.2 SOME-Bus DSM Multiprocessor Model 2

The state of the network shown in Figure 4.2 is completely defined by $\mathbf{K}_{i_j}$, the vector representing the number of chain *j* messages present at service

center *i*. Given exponentially distributed service times for all servers, these states form a Markov Chain. The number of states associated with a single chain is equal to the number of ways the *K* messages of the chain can be partitioned in the C service centers, as shown in Equation 4.10.

$$S_{K,C} = \binom{C-1+K}{K} \tag{4.10}$$

Each chain of this model is assumed to contain three messages. The choice of three messages is based on experimental results indicating unacceptable latency (relative to thread execution time) for responses to remote-memory requests by more than three threads. These results are in agreement with the findings of other researchers studying multithread execution in multiprocessor systems [42], [1]. With *K*=3 and C=4, the number of states for a single chain of the model equals 20, making the total number of states for the two-chain model 400. The state probabilities, used to determine performance measures, are found by solving the system of global balance equations

$$\boldsymbol{\pi Q} = \mathbf{0} \tag{4.11}$$

where $\boldsymbol{\pi}$ is the state-probability vector, and $\mathbf{Q}$ the transition-rate matrix [27]. Equation 4.11 illustrates the fact that probability flux among the states is balanced at equilibrium. The system of Equation 4.11 has one linearly dependent equation that is replaced by the conservation relationship shown in

Equation 4.12 (with S representing the set of states in the Markov chain) to permit a solution.

$$\sum_{s \in S} \pi_s = 1 \tag{4.12}$$

To determine processor center utilization for a chain, let P_{TR} represent the set of states in the Markov chain where a processor center has messages in both queues. Let P_{T0} represent the set of states where responses are enqueued, but no remote-memory requests are present. When both chains are present, threads receive only α_T of the available service, resulting in thread-processing processor utilization of

$$util_T = \alpha_T \cdot \sum_{i \in P_{TR}} \pi(i) + \sum_{j \in P_{T0}} \pi(j) \tag{4.13}$$

for either node. Processor center utilization for remote-memory requests can be found in a similar fashion using the sets P_{TR} and P_{0R}, and the fraction α_R, as seen in Equation 4.14.

$$util_R = \alpha_R \cdot \sum_{i \in P_{TR}} \pi(i) + \sum_{j \in P_{0R}} \pi(j) \tag{4.14}$$

With no service distinction between remote memory requests and responses at a channel service center, channel utilization is determined by summing the

probabilities associated with the set of states in *S* where the channel center is occupied.

The formula for the average number of chain *j* messages at center *i* is shown in Equation 4.15, where $\pi_s(k = k_{i_j})$ are the probabilities associated with the states of the Markov chain where $k = k_{i_j}$.

$$n_{i_j} = \sum_{k=1}^{K}\left(k \cdot \sum_{s \in S} \pi_s(k = k_{i_j}) \right) \tag{4.15}$$

To calculate the average time a message of chain *j* spends in center *i* of Figure 4.2 (queued and in service), the average-time-in-center formula from the MVA method is again applied. At the channel center, all arriving messages are placed in the same queue and receive the same service. Given equal service times $(\tau_{i_0} = \tau_{i_1} = \tau_c,\ i = 2,4)$, the arriving message from chain *r* observes the equilibrium solution of the network with one less customer in its chain [46]. This condition is reflected in the average number of messages of chain *r* present in Equation 4.16.

$$t_{i_r} = \tau_c + \tau_c \cdot (n_{i_r}(K-1) + n_{i_q}(K)) \qquad (q \neq r \text{ and } i = 2,4) \tag{4.16}$$

To determine the average time threads or requests spend at processing center *i* requires redefining thread and request service times, due to the processor sharing described above. This redefined service time is conditioned

on the fact that the processing center must be occupied. During those periods where messages from only one chain are enqueued, the average service time is τ_{i_j}. When messages from both chains are present, the average service time changes to τ_{i_T}/α_T for thread processing, and τ_{i_R}/α_R for requests. The percentages of time each of these situations occur are found using the same sets of states used in the utilization calculations (Equations 4.13 and 4.14). As a result, Equation 4.17 redefines the service time for threads

$$\hat{\tau}_{i_T} = \frac{\left(\tau_{i_T}/\alpha_T\right)\cdot\left(\sum_{j\in P_{TR}}\pi_j\right) + \tau_{i_T}\cdot\left(\sum_{p\in P_{T0}}\pi_p\right)}{\sum_{l\in P_{TR}\cup P_{T0}}\pi_l} \quad (\text{where } i = 1,3 \text{ and } T = \textit{threads}) \quad (4.17)$$

while Equation 4.18 does the same for requests.

$$\hat{\tau}_{i_R} = \frac{\left(\tau_{i_R}/\alpha_R\right)\cdot\left(\sum_{j\in P_{TR}}\pi_j\right) + \tau_{i_R}\cdot\left(\sum_{p\in P_{0R}}\pi_p\right)}{\sum_{l\in P_{TR}\cup P_{0R}}\pi_l} \quad (\text{where } i = 1,3 \text{ and } R = \textit{requests}) \quad (4.18)$$

Substituting the redefined service time for threads stated in Equation 4.17 into the MVA formula for average time at a service center allows calculation of the average time between the arrival of a response message and its associated thread issuing another memory request.

$$t_{i_T} = \hat{\tau}_{i_T} + \hat{\tau}_{i_T} \cdot n_{i_T}(K-1) \qquad \text{(where } i = 1,3 \text{ and } T = \textit{threads}\text{)} \qquad (4.19)$$

Using the redefined service time for remote-memory requests seen in Equation 4.18,

$$t_{i_R} = \hat{\tau}_{i_R} + \hat{\tau}_{i_R} \cdot n_{i_R}(K-1) \qquad \text{(where } i = 1,3 \text{ and } R\text{=}\textit{requests}\text{)} \qquad (4.20)$$

is the average time a remote-memory request will spend at a processing center.

The next chapter analyzes performance results of the two DSM SOME-Bus models developed in this chapter. Comparisons to results of DSM SOME-Bus simulations are performed to provide insight into the accuracy of these approximate models. Performance results from crossbar and torus DSM system simulations are also examined to determine how well the DSM SOME-Bus configuration performs relative to these two popular topologies.

Chapter 5

DISTRIBUTED-SHARED-MEMORY MODEL PERFORMANCE EVALUATION

This chapter presents the results of theoretical and simulation modeling of distributed-shared-memory (DSM) SOME-Bus systems, along with performance comparisons to crossbar and torus simulations. Performance measurements provided by the two theoretical models developed in Chapter 4 are used to evaluate DSM SOME-Bus processor utilization and communication latency and validate two corresponding versions of DSM SOME-Bus simulators. The simulator associated with the model developed in section 4.2.2 of Chapter 4 is extended to perform DSM system activities that are difficult or impossible to model analytically. Results produced by this simulator are then compared to crossbar and torus DSM-system simulations.

Models of similar systems found in the literature seem to assume that DMA activity at a node does not interfere with the ability of a processor to execute threads. As the results presented in this chapter show, this assumption is incorrect due to memory accesses brought about by remote cache misses. The first DSM SOME-Bus model was created under the assumption that maximum interference occurs as a result of remote cache misses and serves as a useful, worst-case performance analysis tool. By modeling DMA activities in the manner that they are performed by currently-available hardware, the second SOME-Bus DSM model provides a method to realistically assess the impact of remote cache misses on processor performance.

5.1 DSM Model 1 Results

This section compares the performance results produced by the SOME-Bus-based DSM multiprocessor theoretical model developed in section 4.2.1 of Chapter 4 to those produced by a corresponding simulator. In this model each node has a single input queue, with the processor at each node receiving and generating two types of messages while executing $K = 3$ threads. All messages to the single input queue are served in a first-come, first-served manner. This approach corresponds to the case of a processor in an actual system having exclusive access to memory resident at its node while a thread executes. The two types of messages are classified as remote-memory requests and responses. Nodes generate a remote-memory request message when a thread, enabled by a response message entering processor service, has a cache miss that cannot be satisfied locally. Nodes generate a response message when a remote-memory request reaches the head of the processor queue (at context switch points), and receives service representing a set of DMA accesses assembling a response message. Service times for the two message types are exponentially distributed and can have different mean values.

Model parameters that were manipulated for this study include average thread processing time, (t_p), average DMA service time, (t_r), and average channel service time (t_c, which is proportional to message size). As mentioned above, there are $K = 3$ threads assigned to each node, and simulations were performed with the number of nodes (N) equal to 64. The average channel and DMA service times are set equal to each other for each model run. This

allows examination of performance results from both the theoretical model and the simulator, over a full range of input values. The primary means of comparing analytical and simulation results involves examining processor utilization (average fraction of time dedicated to thread processing) and communication latency (channel queueing plus transfer time).

The reference point for all timing parameters and measurements is the average thread processing time prior to a cache miss that cannot be satisfied locally. This average is kept at 100 time units for all model runs. Mean channel and DMA service times are varied from 5 to 100 time units. All performance data is presented so that the x-axis of the associated plot represents the ratio of average message transfer time to average thread processing time (t_c/t_p). Destination node selection is uniform over all nodes (excluding the transmitter) in the SOME-Bus simulator.

The ratio of t_c/t_p is in the proper range to reflect a miss rate of approximately 10%. This was determined by letting m be the miss rate and f the number of instructions-per-second executed by the processor at each node. In addition, let M be the mean message size in bytes and C the channel bandwidth (in bytes per second). As a result, the mean thread runtime, t_p, is equal to $1/(mf)$, and the mean message transfer time, t_c, is equal to M/C, making $t_c/t_p = mfM/C$. In current massively-parallel systems, the ratio of f/C is in the range of 0.5 to 1. For example, the Cray T3D can execute 150×10^6 instructions-per-second, and its byte-wide network links operate at 150MHz. The Cray T3E has the same processing capability, but its network links have 2 to 4 times more bandwidth. In the ASCI RED, each node has two processors with peak-

processing capability of $200x10^6$ instructions-per-second, and bi-directional network links capable of 400 Mbytes per second in each direction. These values translate into a t_c/t_p ratio of approximately .05 to 1 for small cache blocks and miss rates of 10 percent or less.

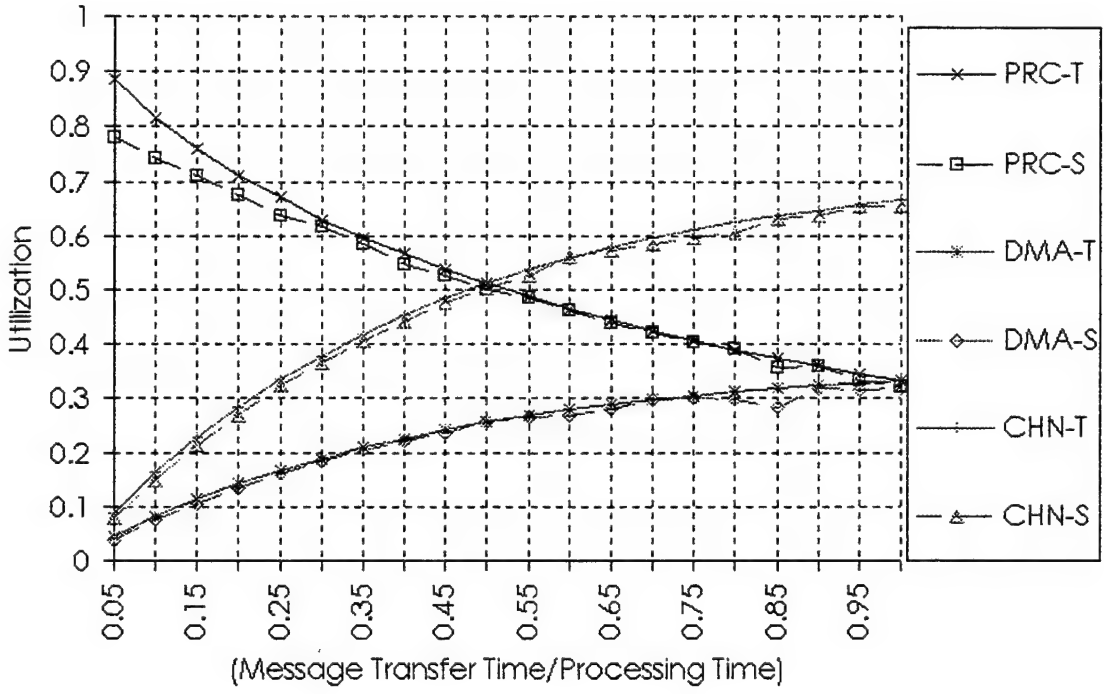


Figure 5.1 SOME-Bus DSM Model 1 Processor, DMA, and Channel Utilization

Figure 5.1 shows average utilization of various components in the SOME-Bus system, as produced by the theoretical (-T) and simulator (-S) models. These include the processor (PRC), DMA (DMA), and channel (CHN). Note that there is excellent correspondence between theoretical and simulation results over the majority of the t_c/t_p range (which is also the t_r/t_p range, since t_r is set equal to t_c). When short responses to remote memory requests are assembled and

transmitted, processor utilization is approximately 90 percent. However, under the single-input-queue premise of this model, as the DMA service time increases in response to longer remote-request messages, the processor is blocked from executing threads, and utilization drops. Note that channel utilization remains below 70 percent over the entire t_c/t_p range, showing a significant reserve of channel capacity to service higher miss rates by the processor.

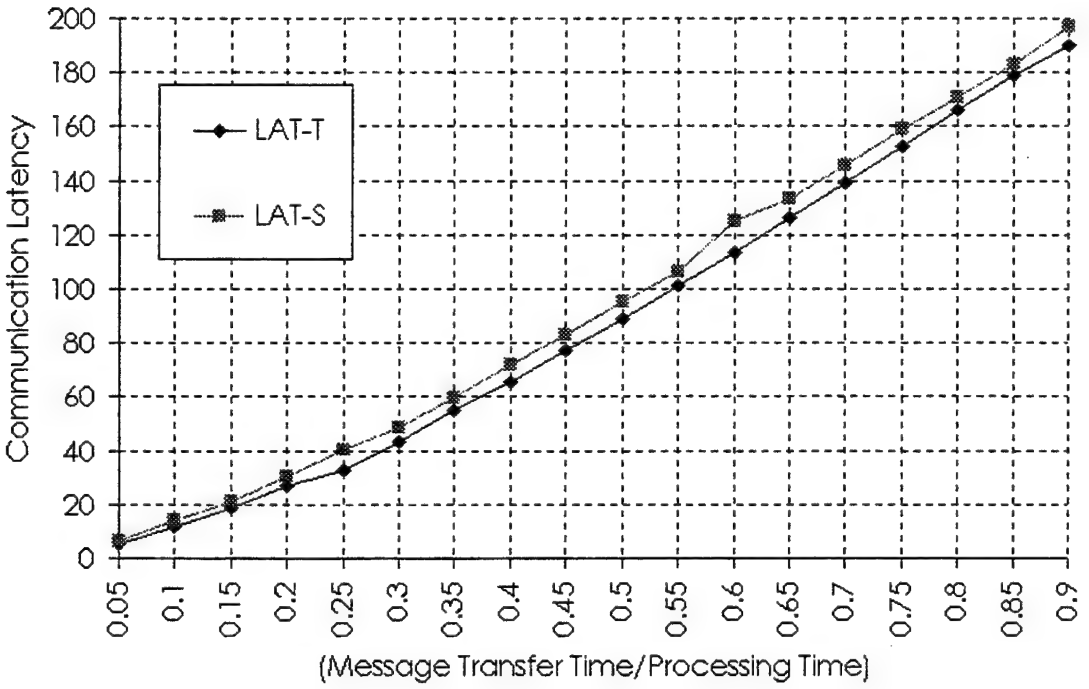


Figure 5.2 SOME-Bus DSM Model 1 Communication Latency

Communication latency (LAT: consists of waiting plus channel transfer time) for the first DSM SOME-Bus model is seen in Figure 5.2. Again, there is close correspondence between theoretical (-T) and simulator (-S) results. Note the almost linear growth in latency as remote-memory and response messages grow

proportionally to channel transfer time. This scalability is due to the constant channel bandwidth available in the SOME-Bus, a system with no contention for switching or routing hardware.

As the results of the next section show, the single-input-queue processor architecture does not perform as well as the two-queue processor sharing method examined using DSM SOME-Bus model 2. In addition, the second model presents a more realistic view of processing node actions, given the current state of processing hardware. These facts make comparisons of this model to torus and crossbar architectures unnecessary.

5.2 DSM Model 2 Results

This section presents performance results from the second theoretical model of a DSM SOME-Bus system, and uses them to validate the performance of a corresponding SOME-Bus simulator. Once validated, the simulator is extended to perform commonly occurring DSM system tasks that are difficult, or impossible, to model analytically. Results from this extension to the simulator are then compared to torus and crossbar DSM simulations. The distinction between this model and the one studied in the previous section is that the input channel queue at each node logically appears as two queues. As messages arrive at a node, remote-memory request messages enter one queue, while response messages enter the other. The messages at the heads of the two queues share service from the local memory which, in effect, becomes the server. The fraction of service α_T goes to response messages (for thread processing), and the

fraction α_R goes to remote-memory requests (for DMA processing). The two fractions sum to one.

Simulators for all three architectures contain $N = 64$ nodes, with each node containing a receiver capable of receiving messages simultaneously on its input channel(s), a processor with cache, a portion of global memory, and an output channel. Multithreaded processing is assumed, with each processor assigned a set of $K = 3$ parallel threads. Thread processing in all simulators consists of a thread executing until it encounters a cache miss that requires remote data. At that point the thread suspends until a response message (modeling the requested data) is received from a remote node. The arrival of the response message enables the thread for execution, and it resumes execution when that message reaches the head of its queue.

When a thread has a cache miss, as described above, a remote-memory request message is enqueued for transmission on the output channel. After transfer time t_c expires, the message is enqueued at the receiver of the destination (remote) node to await service by the DMA processor. The DMA processor requires time t_r to assemble the response message and enqueue it for transmission on the output channel of the remote node. The response message is then returned to the originating node.

In DSM systems, write accesses to global memory generate invalidation (or update) messages that must be transmitted on the interconnection network. Unlike the torus and the crossbar architectures, this additional traffic has no effect on SOME-Bus network communication latency (waiting plus channel transfer time) since there is no contention for shared communication resources

(thus no potential for blocking). However, this traffic does have an effect at the receiving processor in all architectures. The model for issuing invalidation messages, used for all three architectures, assumes the source node has a directory of the nodes with cached copies of the affected data.

In all three simulators, when invalidation messages are modeled they accompany remote-memory requests. In the SOME-Bus, a single invalidation message is broadcast, with a list of recipient nodes. In the crossbar, a node waits until input channels to all nodes with copies of the invalid data are available, reserves them, and then broadcasts a single invalidation message to those destinations. In the torus, a spanning tree is created from the source node and multi-destination worms broadcast the invalidation message. The number of nodes receiving invalidation messages (0 or 10), and the amount of processing time those messages receive (5 time units) are parameters of the simulation. Although invalidation messages receive service at the destination node, no response is transmitted back to the source node.

The major parameters of this study are the distribution type and average length of thread and DMA service times, distribution type and average length of messages, and the destination node selection distribution. Quantities used to measure and compare the performance of the architectures include the average utilization of various system components (fraction of busy time) and the average communication latency (channel waiting plus transfer time) that a message experiences.

The reference point for the timing parameters and measurements of the study is the thread runtime, t_p, geometrically distributed with a mean of 100 time

units. Message transfer time, t_c, is geometrically distributed with a mean in the range from 3 to 100 time units, placing t_c/t_p in the range of 0.03 to 1, for the same reasons described in the previous section. Destination node selection for remote-memory requests in the SOME-Bus and crossbar is uniform over all nodes (excluding the source) and uniform in each direction in the torus. Response messages are returned directly to the requester. The fractions of service assigned to thread and DMA processing are kept equal for all simulation runs ($\alpha_T=\alpha_R=0.5$). This choice was made when initial studies indicated that all three architectures are insensitive to the fraction of memory bandwidth allocated to the DMA controller, unless it is below 15 percent. Establishing paths (or wormholes) to transfer messages in the crossbar and torus occurs in the same manner as that described in Chapter 3, section 3.2.

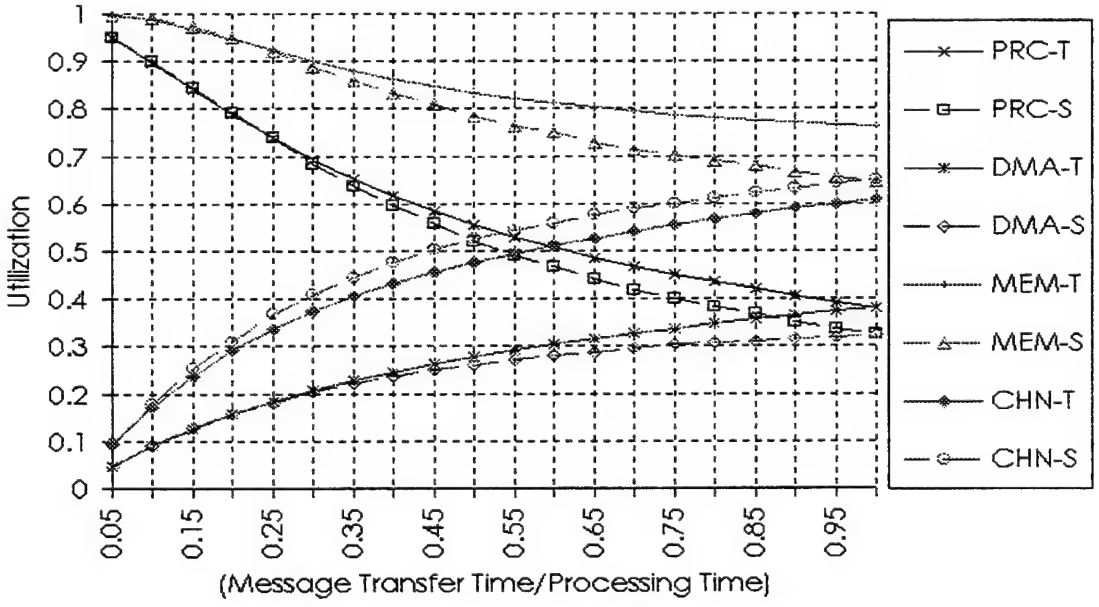


Figure 5.3 DSM SOME-Bus Model 2, Subsystem Utilization

Figure 5.3 compares the average utilization for processor (PRC), DMA (DMA), channel (CHN), and memory (MEM) subsystems produced by the second simulation (-S) and theoretical (-T) DSM SOME-Bus models. For this comparison the DMA processing time (t_r) is again set equal to message transfer time (t_c). As mentioned above, the memory subsystem functions as the server when processor-sharing between threads and remote-memory requests occurs. As a result, the memory utilization curve shown is the sum of processor and DMA utilization.

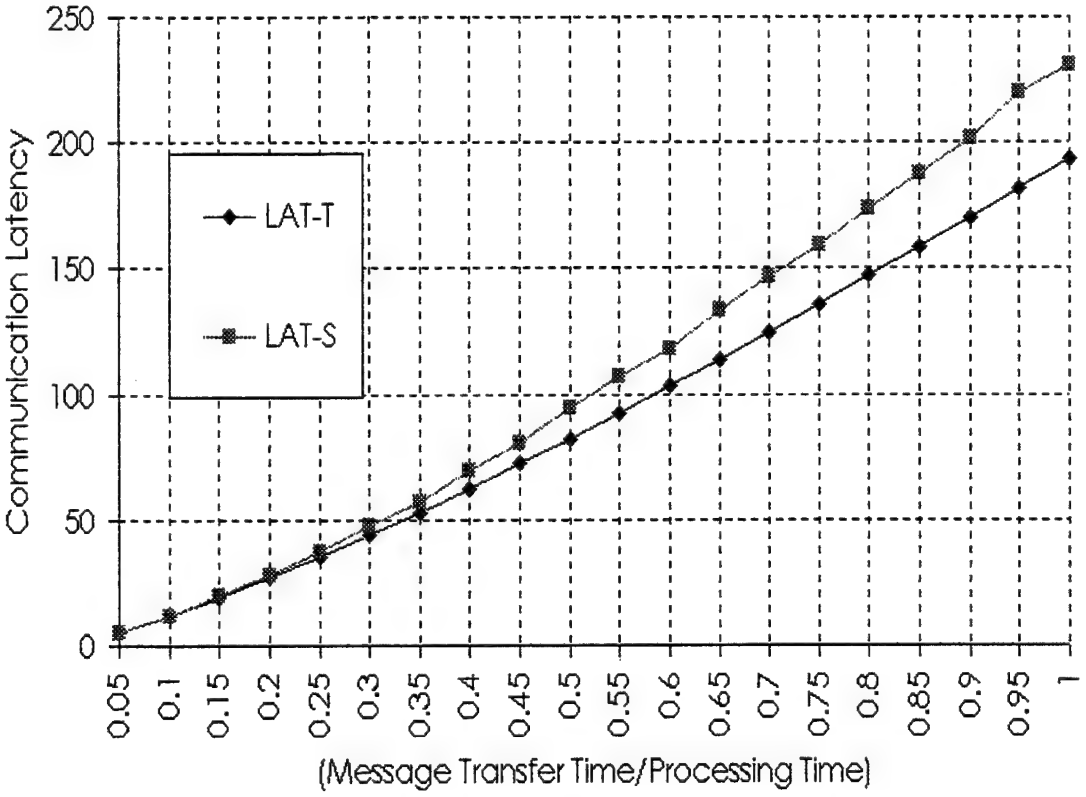


Figure 5.4 DSM SOME-Bus Model 2 Communication Latency

Given the fact that the theoretical model is only an approximation of system behavior, as explained in Chapter 4, it produces results that are very close in value, and identical in trend, to the simulations. With both operations having equal access to memory ($\alpha_T = \alpha_R = 0.5$), memory utilization shows a decline as the average DMA processing time approaches the time to process a thread. Compared to results from the previous model, the processor is better utilized over the entire range of t_c/t_p. Figure 5.4 shows communication latency (channel waiting plus transfer time) for this model, again indicating an almost linear increase in response to longer messages and DMA processing times.

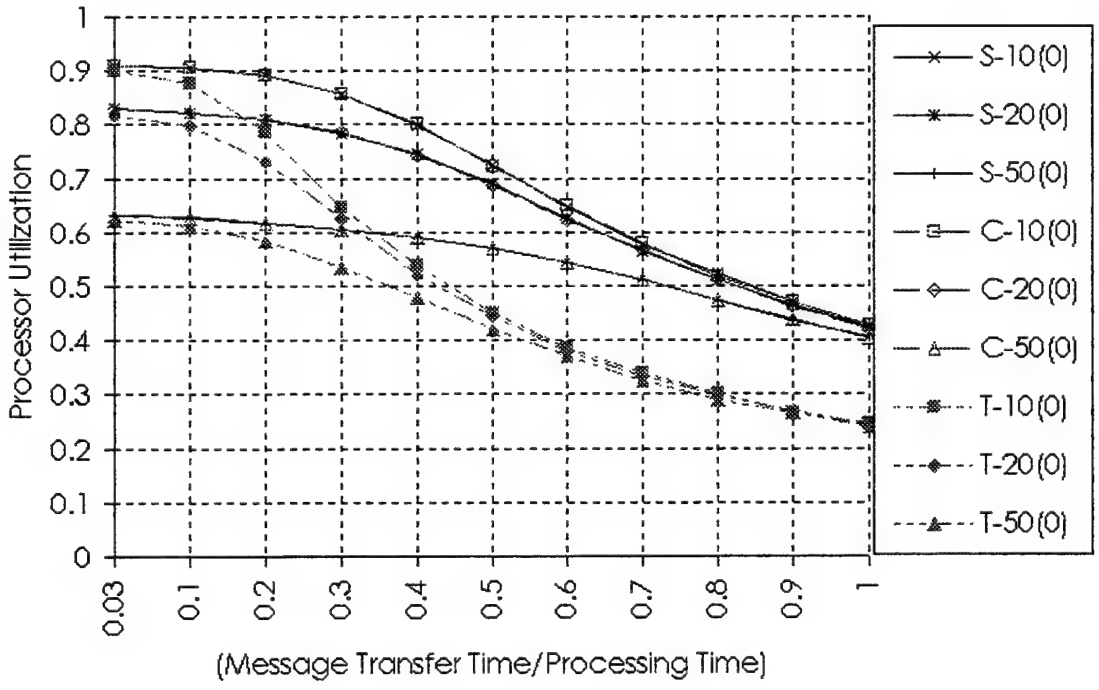


Figure 5.5 All DSM Architectures, Processor Utilization, No Invalidation Messages

The results shown in Figures 5.3 and 5.4 indicate that the simulator accurately reflects the second DSM SOME-Bus theoretical model performance predictions. Extending this simulator to incorporate the issuance of cache invalidation (update) messages, as explained above, provides a basis for realistic comparisons to torus and crossbar DSM architectures. Figure 5.5 shows processor utilization for the SOME-Bus (S), crossbar (C), and torus (T) as DMA processing times are varied between 10 (-10), 20 (-20), and 50 (-50) time units. The baseline results, produced when no cache invalidation messages (0) are transmitted, show SOME-Bus and crossbar processor utilization curves are almost identical, while the torus exhibits a rapid decline as message lengths increase. As expected with all architectures, processor utilization declines as average DMA processing time or message length increases.

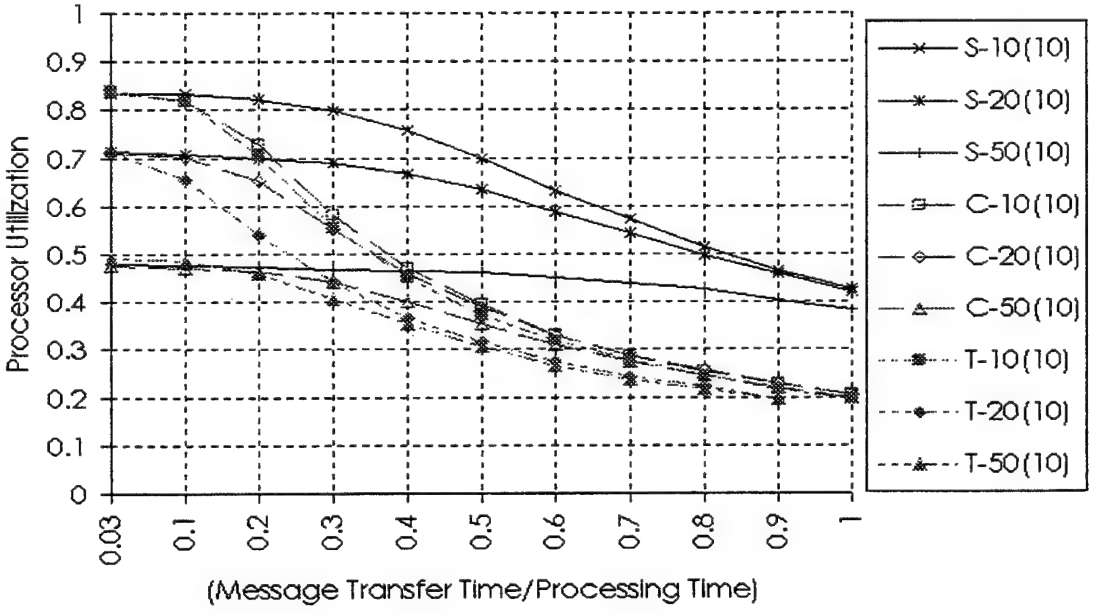


Figure 5.6 All DSM Architectures, Processor Utilization, 10 Invalidation Messages

Figure 5.6 shows the effect of invalidation message traffic, and its associated processing, on processor utilization when 10 nodes (10) receive the invalidation message that accompanies each remote-memory request. The crossbar now begins to react like the torus as it waits for communication channels to become available for an invalidation-message broadcast. Except for a constant offset, SOME-Bus performance here is identical to that shown in Figure 5.5, the loss due only to invalidation-message processing, not network characteristics.

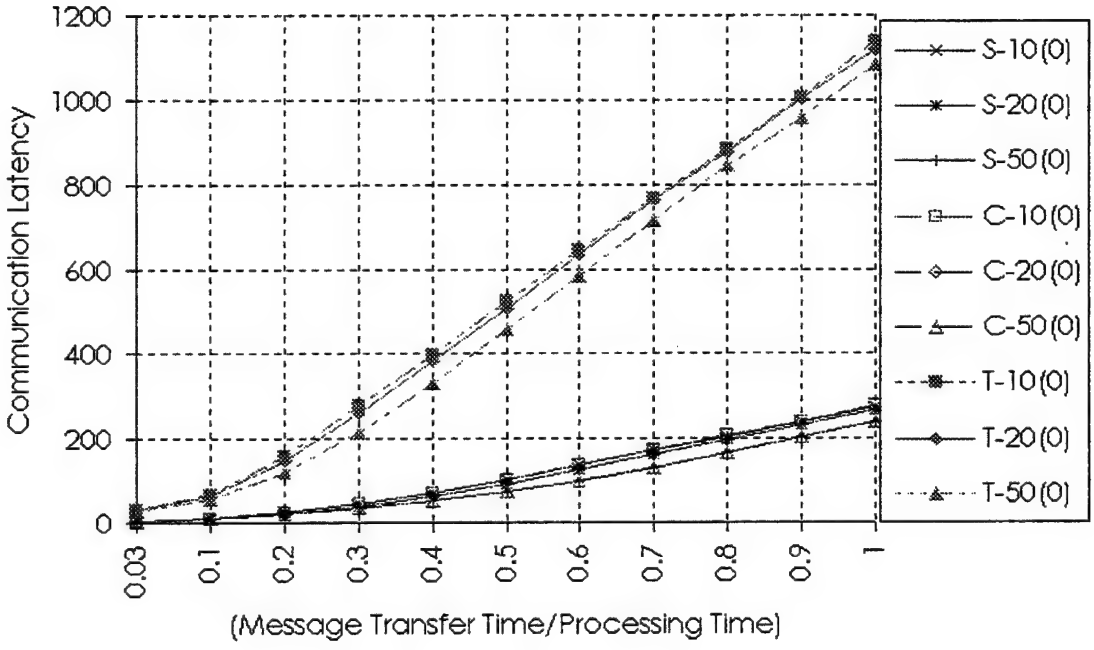


Figure 5.7 All DSM Architectures, Comm Latency, No Invalidation Messages

Communication latency for all three architectures, for the case of no invalidation messages and 10, 20, and 50 time unit average DMA processing

times, is seen in Figure 5.7. Again, the SOME-Bus and crossbar show almost identical performance, while torus latency increases rapidly in response to increasing message length. Examination of additional simulation statistics show this rise is in direct relationship to the number of blocked wormholes.

Figure 5.8 shows communication latency in all three architectures when cache invalidation messages are transmitted. Comparing this figure to Figure 5.7 shows that the additional transmissions essentially have no effect on SOME-Bus communications, while torus and crossbar systems are both affected. Crossbar latency grows relative to the availability of channels connected to the nodes receiving invalidation messages. As message lengths increase, the associated wait for channels needed for invalidation-message broadcasts also increases, with the final outcome being increased latency.

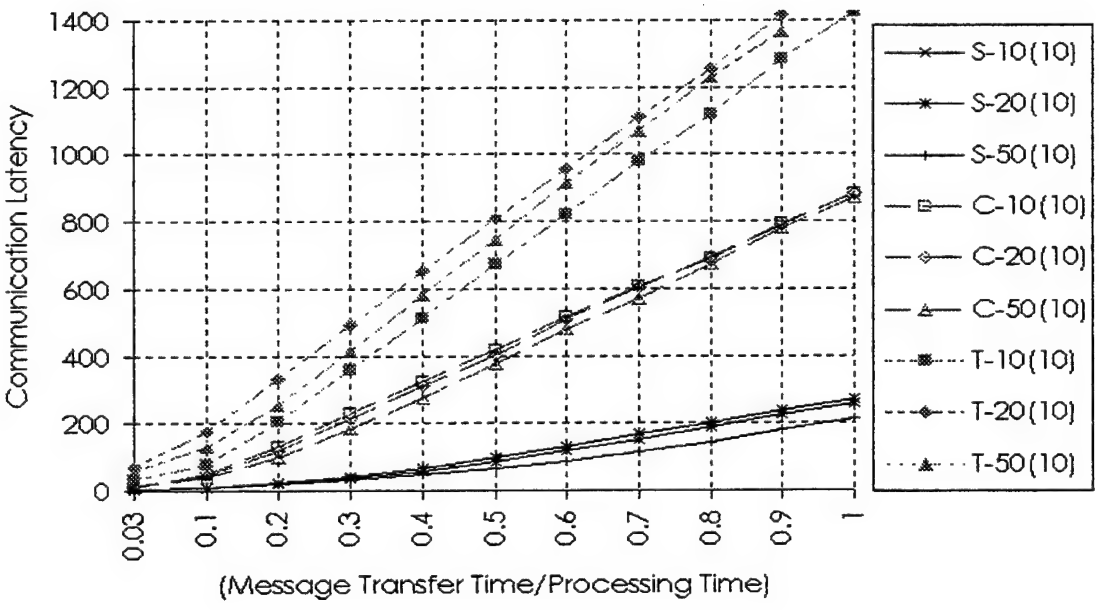


Figure 5.8 All DSM Architectures, Comm Latency, 10 Invalidation Messages

Figures 5.9 and 5.10 show channel utilization curves associated with the communication latencies shown in Figures 5.7 and 5.8. The torus quickly reaches a plateau of channel utilization, whether invalidation messages are broadcast or not, as channels needed by one wormhole are blocked by another. Figure 5.9 shows that, In the absence of invalidation messages, the SOME-Bus and crossbar exhibit almost identical performance. However, when invalidation messages are broadcast, as shown in Figure 5.10, the crossbar begins to react in a manner similar to the torus, waiting for blocked channels to become available so invalidation-message broadcasts can take place. Figure 5.10 also shows a rise in the average channel utilization rate of the SOME-Bus, but the curves do not show channel saturation, even when messages reach the maximum length studied.

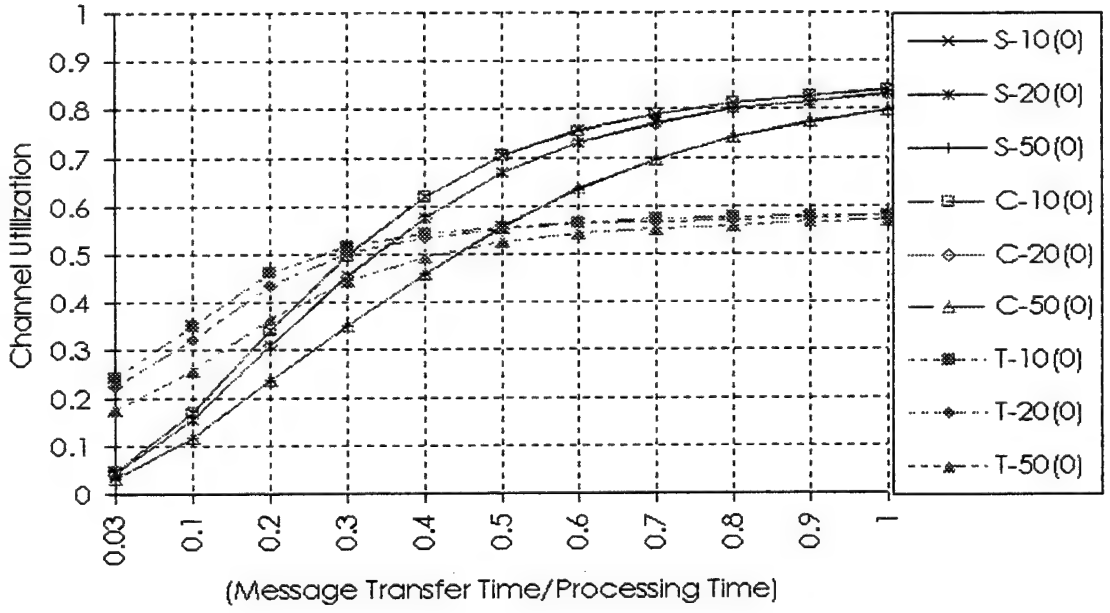


Figure 5.9 All DSM Architectures, Channel Utilization, No Invalidation Messages

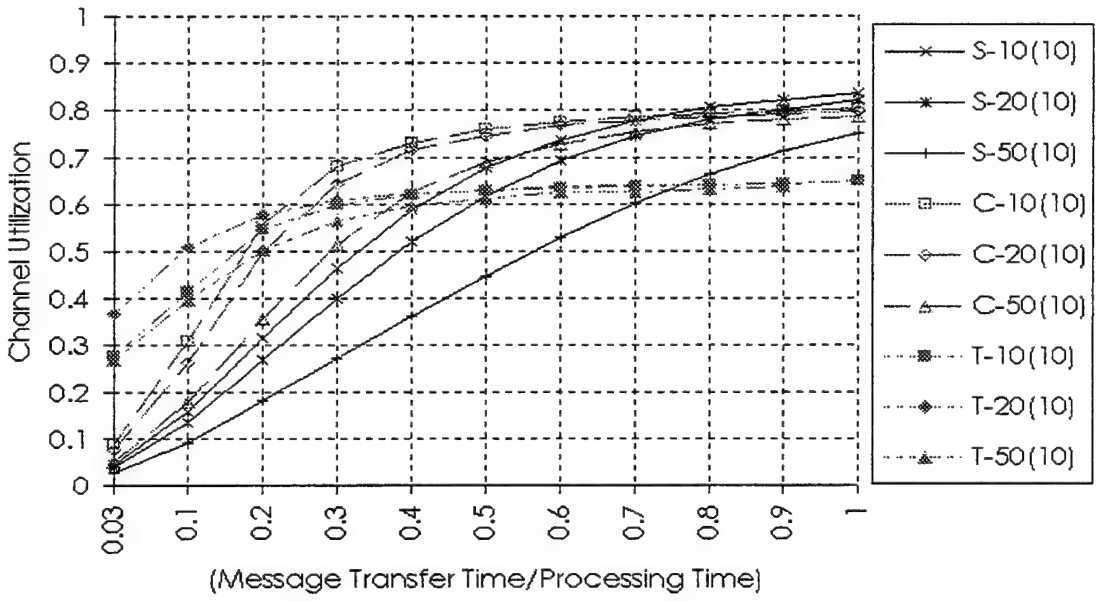


Figure 5.10 All DSM Architectures, Channel Utilization, 10 Invalidation Messages

An additional study of this (second) model of DSM architectures was performed with $K = 1$ threads assigned to each node. In this situation, all architectures were found to have similar processor utilization simply because all the networks were able to effectively deliver, without significant delay, the reduced number of messages that resulted from the execution of fewer threads. The more realistic case of three threads-per-processor, studied in this chapter, indicates that there are pronounced differences in performance between the three architectures in the presence of significant message traffic on the network.

Chapter 6

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

The execution of high-performance applications is now a common occurrence in the engineering, science, and business communities. Users want to execute larger or more detailed models in less time and now accept parallel processing systems as a useful tool for performing this work. The key to exploiting the parallelism in these applications is efficient communication between task processes, which implies a corresponding need for efficient communication among parallel-system processors. The Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus) meets this need with its low-latency, high-bandwidth, broadcast-based fiber-optic interconnection network that is able to interconnect over one hundred processor nodes, directly linking arbitrary pairs without contention.

An examination of existing multicomputer and multiprocessor interconnection-network characteristics, both static and dynamic, provided a quantitative basis for determining SOME-Bus performance potential. The results of these comparisons indicated that the SOME-Bus exhibits outstanding performance potential. Compared with static topologies, a SOME-Bus offers the same performance as the powerful yet costly fully-connected network, but at a lower cost than a hypercube, mesh, or torus. Contrasted with the most versatile dynamic network, the crossbar, the SOME-Bus is scaleable and directly supports

more communication patterns. In addition, due to the optical implementation of its broadcast-based design, no node is ever blocked from transmitting by another node, and no arbitration for shared resources is required. Even with the SOME-Bus requiring N^2 receivers, larger than the number required in other architectures, the hardware scales by O(N), since N receivers are integrated into a single device at relatively low cost.

Research studies show that common high-performance applications include modeling and simulation of physical phenomena, integrated circuits, neural networks, weather, economic systems, and image processing. Some of the basic mathematical operations used to execute this workload include fast Fourier transforms (FFTs), matrix multiplication, Gaussian elimination, LU-factorization, and solutions to partial differential equations. Parallel-processing implementations of these operations typically involve collective communications, including one-to-all and all-to-all broadcasts of data and synchronization information. On existing parallel processing systems, processor performance declines as these collective-communication operations result in increased communication latency. Results of numerous studies indicate that the high-performance applications cited above execute on existing parallel systems at only poor-to-moderate performance levels, even after extensive efforts at software tuning. Based on these findings, it is apparent that the SOME-Bus, a scaleable parallel system that directly supports the collective communication operations inherent in common high-performance applications, has the potential to outperform existing systems executing those same applications.

A theoretical examination of the SOME-Bus as a message-passing system was performed using a closed, single-class queueing network model. An efficient solution method for the product-form equation associated with this network was developed based on an iterative application of Norton's Theorem for Queueing Networks. This solution method permits the evaluation of hot-spot performance and allows the use of any model of a node that can be represented by a product-form queueing network. Performance measurements provided by the model were used to evaluate processor utilization and communication latency, as well as validate a SOME-Bus simulator with the same average service times and distributions. The simulator was extended to perform synchronization operations and exchanges of intermediate data results that cannot be modeled analytically. Results from the modified SOME-Bus simulator were compared to crossbar and torus simulations to determine the relative performance of the SOME-Bus architecture with respect to these two network topologies.

As a message-passing system, simulation results (for a 64-node system executing 192 total tasks with no synchronization operations) indicate that the SOME-Bus, crossbar, and torus display similar performance (processor utilization) for coarse- to medium-grain parallel processes. However, as message transfer time approaches computation time, processor utilization in the torus decreased at a markedly greater rate than the other two architectures. With respect to communication latency (message waiting plus channel transfer time), the SOME-Bus and crossbar systems exhibited identical performance, while the torus system experienced significant additional delays as message sizes were increased.

In the presence of both heavy (frequent) and light (infrequent) synchronization operations, the SOME-Bus outperforms the other two architectures, seemingly unaffected by the exchange of intermediate data results prior to a synchronization operation. In contrast, heavy synchronization operations, coupled with message transfer times that approached computation times, appear to have a dramatic effect on the ability of torus and crossbar systems to keep their processors occupied with assigned tasks. With regard to communication latency in the presence of synchronization, SOME-Bus results were identical to those generated when no synchronization took place. In comparison, the crossbar was significantly affected by heavy synchronization operations, and even light synchronization noticeably increased latency (with respect to no synchronization). The torus had at least twice the latency, on average, as the SOME-Bus, regardless of the type of synchronization that occurs. The SOME-Bus clearly has less latency, in all synchronization situations, than the crossbar or the torus. As a message-passing parallel processing system, the SOME-Bus clearly outperformed these two architectures when synchronization operations occurred, or when the processing tasks were fine-grained.

Two theoretical models were developed and used to evaluate the SOME-Bus as a distributed-shared-memory (DSM) parallel processing system. By modeling both processor and DMA activity at a node, the contention for memory that is introduced by remote-memory requests becomes apparent. This approach to DSM modeling is contrary to most models of similar systems found in the literature; those models seem to assume that DMA activity at a node does

not interfere with the ability of a processor to execute threads. As the performance results showed, this activity should not be ignored.

The first DSM SOME-Bus model was created by assuming that the maximum interference possible occurs as a result of remote cache misses; it serves as a useful, worst-case performance analysis tool. The second SOME-Bus DSM model treats DMA activities in the manner that they are performed by commercial-off-the-shelf hardware, thus providing a realistic assessment of the impact of remote cache misses on processor performance. Again, the theoretical model was used to validate a baseline simulator, which was then extended to incorporate cache-coherence operations that cannot be modeled analytically. Results from the modified simulator were then compared to DSM torus and crossbar simulations.

Results from the first model show that when short responses to remote-memory requests are assembled and transmitted, processor utilization in a 64-node system executing 3 tasks-per-node is approximately 90 percent. However, under the single-input-queue premise of this model, as DMA service time increased in response to longer remote-request messages, memory interference blocked the processor from executing threads and utilization dropped. Channel utilization, however, remained below 70 percent over the entire range of DMA and channel transfer times studied, which indicates a significant reserve of channel capacity to service higher miss rates by the processor. There was almost linear growth in communication latency as the time to transfer remote-memory and response messages increased. This characteristic illustrated the

availability of constant channel bandwidth in the SOME-Bus, a system with no contention for switching or routing hardware.

By extending the simulator associated with the second DSM SOME-Bus model to incorporate the issuance of cache invalidation (update) messages to accompany remote-memory requests, the basis for realistic comparisons to torus and crossbar DSM architectures was created. With DMA and thread processors sharing equal access to memory at a node, processor utilization was examined as DMA processing and channel transfer times were varied. The baseline results, produced when no cache invalidation messages were transmitted, show SOME-Bus and crossbar processor utilization curves are almost identical, while the torus exhibits a rapid decline relative to the others as message length increased. Regarding communication latency, the SOME-Bus and crossbar again show almost identical performance, while latency in the torus increased rapidly in response to increasing message length. This increase was in direct relationship to the number of blocked wormholes in the system.

When cache-invalidation messages accompanied remote-memory requests, only a minor reduction in processor utilization was seen in the SOME-Bus, and its communication latency was unaffected. The reduction in processor utilization was attributed to invalidation-message processing, not network characteristics. Crossbar and torus processor utilization declined rapidly as message lengths increased and communication channels became blocked by invalidation-message broadcasts. Crossbar and torus latency grew relative to the availability of channels connected to the nodes receiving invalidation

messages. As message lengths increased, the associated wait for the set of channels to broadcast invalidation-messages also increased.

A contributing factor for the increased latency of the torus and crossbar systems was found in an examination of channel utilization. The torus quickly reached a plateau of channel utilization, whether invalidation messages were broadcast or not, as channels needed by one wormhole were blocked by another. When invalidation messages were broadcast, the crossbar began to react in the same manner, as it waited for blocked channels to become available for invalidation-message broadcasts. Channel saturation was clearly evident for the torus and crossbar systems. In the final analysis, when cache-coherence is considered, the SOME-Bus clearly outperformed the crossbar and torus as a DSM system.

When the performance results presented in this dissertation are considered along with the advances in optical technology, it is readily apparent that the SOME-Bus is a viable parallel processing system. Its performance derives from the fact that no routing or arbitration is required for communications, allowing the system to function as though it were a fully-connected network. The hardware and communication bandwidth scale directly, and as optical components mature, available bandwidth will increase. Even the modest implementations described in Chapter 1 have bandwidths comparable to other state-of-the-art architectures. With receiver logic designed to efficiently support programming models that are commonly implemented on parallel computers, the SOME-Bus greatly simplifies parallel programming.

6.2 Future Directions

Direct extensions to the work presented in this dissertation include the evaluation of a message-passing SOME-Bus with different processing architectures and hot-spot conditions. Also, by extending the application of Norton's Theorem for Queueing Networks presented in Chapter 2 to include multi-class queueing networks, other models become feasible. First, by using a more complex model of a node, in combination with the network configuration presented in Chapter 2, an exact DSM model corresponding to the second model presented in Chapter 4 may be possible. Second, an examination of the SOME-Bus as a central-shared-memory multiprocessor, where memory-request messages issued by processing nodes form one customer class and responses to those requests by the memory subsystem forms another, could be constructed using relatively simple models of processing and memory nodes. Finally, different receiver logic designs could be evaluated by creating a more complex model of a node that included separate service centers to represent the different activities that take place within the receiver.

REFERENCES

[1] Adve, V.S., and M.K. Vernon, 1994. "Performance Analysis of Mesh Interconnected Networks with Deterministic Routing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 3, pp 225-246.

[2] Agarwala, A., and C.R. Das, 1995. "Experimenting with a Shared Virtual Memory Environment for Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 29, No. 2, pp. 228-235.

[3] Baskett, F., K.M. Chandy, R.R. Muntz, and F.G. Palacios, 1975. "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *Journal of the ACM*, Vol. 22, No. 2, pp. 248-260.

[4] Bell, G. 1992. "Ultracomputers, A Teraflop Before Its Time," *Communications of the ACM*, Vol. 35, No. 8, pp 27-47.

[5] Bhuyan, L.N., R.R. Iyer, and M. Kumar,1997. "Performance of Multistage Bus Networks for a Distributed Shared Memory Multiprocessor," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 1, pp. 82-95.

[6] Bogineni, K., and P.W. Dowd, 1992. "Performance Analysis of Two Address Space Allocation Schemes for an Optically Interconnected Distributed Shared Memory System," *Proceedings of the International Conference on Parallel Processing*, pp. 562-566.

[7] Bruell, S.C., and G. Balbo, 1980. *Computational Algorithms for Closed Queueing Networks*. North Holland.

[8] Buzen, J.P., 1973. "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Communications of the ACM*, Vol. 16, No. 9, pp. 527-531.

[9] Calvin, C. 1995. "All-To-All Broadcast in Torus with Wormhole-Like Routing," *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, pp. 130-137.

[10] Camp, W.J., S.J. Plimpton, B.A. Hendrickson, R.W. Leland. 1994. "Massively Parallel Methods for Engineering and Science Problems," *Communications of the ACM*, Vol. 37, No. 4, pp. 31-41.

[11] Chandy, K.M., U. Herzog, and L. Woo, 1975. "Parametric Analysis of Queuing Networks," *IBM Journal of Research and Development*, Vol. 19, No. 1, pp. 36-42.

[12] Dahlgren, F. and P. Stenstrom, 1996. "Evaluation of Hardware-Based Stride and Sequential Prefetching in Shared-Memory Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 4, pp 385-394.

[13] Demmel, J. 1996. "Performance of a Parallel Global Atmospheric Chemical Tracer Model," *Supercomputing* 96.

[14] Evangelinos, C. 1995. "Communication Performance Models in Prism: a Spectral Element-Fourier Parallel Navier-Stokes Solver," *Supercomputing* 95.

[15] Gordon, W.J., and G.F. Newell, 1967. "Closed Queueing Systems with Exponential Servers," *Operations Research*, Vol. 15, pp. 254-265.

[16] Grujic, A., M. Tomasevic, and V. Milutinovic, 1996. "A Simulation Study of Hardware-Oriented DSM Approaches," *IEEE Parallel and Distributed Technology*, Vol. 4, No. 1, pp. 74-83.

[17] Hendrickson, B., and S. Plimpton, 1995. "Parallel Many-Body Simulations Without All-To-All Communication," *Journal of Parallel and Distributed Computing*, Vol. 27, No. 1, pp. 15-25.

[18] Hines, W.W., and D.C. Montgomery, 1990. *Probability and Statistics in Engineering and Management Science*. Wiley.

[19] Hinrichs, S. 1994. "An Architecture for Optimal All-To-All Personalized Communication," *Carnegie-Mellon Tech. Report* (Cs Cmu-Cs-94-140).

[20] Ho, C. 1995. "Optimal Broadcast in All-Port Wormhole-Routed Hypercubes", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6 No. 2, pp. 203-207.

[21] Huang, C., and P.K. McKinley, 1994. "Communication Needs of Parallel Applications," *IEEE Parallel & Distributed Technology*, Winter, pp. 78-79.

[22] Hwang, K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill.

[23] Johnsson, S. 1989. "Optimum Broadcasting and Personalized Communications in Hypercubes," *IEEE Transactions on Computers*, Vol. 38, pp. 1249-1267.

[24] Katsinis, C., 1998. "Distributed-Shared-Memory Support on the Simultaneous Optical Multiprocessor Exchange Bus," *MASCOTS Conference*, July 21, 1998.

[25] Katsinis, C., 1998. "Performance Analysis and Simulation of the SOME-Bus Architecture Using Message Passing," *Seventh International Conference on Computer Communications and Networks*, October 12, 1998.

[26] Katsinis, C., W. E. Cohen, R. K. Gaede and J. H. Kulick. 1997. "The Architecture and Performance of the Simultaneous Optical Multiprocessor Exchange Interconnection Network (SOME-Bus)," *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications.*

[27] Kleinrock, L., 1975. *Queueing Systems, Vol. I: Theory*. Wiley-Interscience.

[28] Kulick J. H., W. E. Cohen, C. Katsinis, E. Wells, A. Thomsen, R. K. Gaede, R. G. Lindquist, G. P. Nordin, M. Abushagur, and D. Shen. 1995. "The Simultaneous Optical Multiprocessor Exchange Bus," *Proceedings of the Second International Conference on Massively Parallel Processor Optical Interconnects*, pp. 336-344.

[29] Kumar, V., A. Grama, A. Gupta, and G. Karypis. 1994. *Introduction to Parallel Computing, Design and Analysis of Algorithms*. Benjamin/Cummings.

[30] Lan, Y. 1994. "Multicast Communication in 2-D Mesh Network," *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, pp. 63-68.

[31] Lavenberg, S.S., and M. Reiser, 1980. "Stationary State Probabilities at Arrival Instants for Closed Queueing Networks with Multiple Types of Customers," *Journal of Applied Probabiltiy*, Vol. 17, pp. 1048-1061.

[32] Lazowska, E.D., J. Zahorjan, G.S. Graham, and K.C. Sevcik, 1984. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall.

[33] Li, Y., A.W. Lohmann, Z.G. Pan, S.B. Rao, I. Redmond, and T. Wang, 1994. "Optical Multiple-Access Mesh-Connected Bus Interconnections," *IEEE Proceedings*, Vol. 82, No. 11, pp. 1690-1700.

[34] Lin, X. 1993. "Multicast Communications in Multicomputer Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 10, pp. 1105-1117.

[35] Lindquist R.G., J. H. Kulick, W. E. Cohen, R. K. Gaede, E. Wells, M. Abushagur, D. Shen, C. Katsinis, and S. T. Kowel. 1997. "An Optoelectronic Design of the Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus)," *SPIE Proceedings*.

[36] Little, J.D.C., 1961. "A Proof of the Queueing Formula L=λW," *Operations Research*, Vol. 9, pp. 383-387.

[37] Lou, J. 1995. "Performance Analysis and Optimization on the UCLA Parallel Atmospheric General Circulation Model Code", *Supercomputing 95*.

[38] Mabbs, S.A., and K.E. Forward, 1994. "Performance Analysis of MR-1, a Clustered Shared-Memory Multiprocessor," *Journal of Parallel and Distributed Computing*, Vol. 20, No. 2, pp. 158-175.

[39] Marsan, M. A., G. Balbo, and G. Conte, 1986. *Performance Models of Multiprocessor Systems*. MIT Press.

[40] Mckinley, P. 1994. "Unicast-Based Multicast Communication in Wormhole-Routed Network," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 12, pp. 1252-1265.

[41] Mckinley, P. 1995. "Collective Communication in Wormhole-Routed Massively Parallel Computing," *IEEE Computer*, Vol. 28, No. 12, pp. 39-50.

[42] Nemawarkar, S. S., R. Govindarajan, G.R. Gao, and V.K. Agarwal, 1993. "Analysis of multithreaded multiprocessors with distributed shared memory," *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, pp. 114-121.

[43] Nowatzyk, A. 1995. "Are Crossbars Really Dead? The Case for Optical Multiuprocessor Interconnection Systems," *Computing Architecture News*, Vol. 23, No. 2, p. 106.

[44] Patterson, D.A., and J.L. Hennessy, 1996. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann.

[45] Plimpton, S. 1996. "Transient Dynamics Simulations: Parallel Algorithms for Contact Detection and Smoothed Particle Hydrodynamics," *Supercomputing* 96.

[46] Reiser, M., 1979. "A Queueing Network Analysis of Computer Communication Networks with Window Flow Control," *IEEE Transactions on Communications*, Vol. 27, No. 8, pp. 1199-1209.

[47] Reiser, M., and S.S. Lavenberg, 1980. "Mean-Value Analysis of Closed Multichain Queuing Networks," *Journal of the ACM*, Vol. 27, No. 2, pp. 313-322.

[48] Robertazzi, T.G., 1994. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Springer-Verlag.

[49] Scott, S.L., and G.M. Thorson, 1996. "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus," *Proceedings of HOT Interconnects IV*.

[50] Stenstrom, P., and F. Dahlgren. 1996. "Applications for Shared Memory Multiprocessors," *IEEE Computer*, Vol. 29, No. 12, pp. 29-31.

[51] Sterling, T., P. Merkey, and D. Savarese, 1996. "Improving Application Performance on the HP/Convex Exemplar," *IEEEComputer*, Vol. 29, No. 12, pp. 50-55.

[52] Stone, H.S., 1987. *High-Performance Computer Architecture*. Addison-Wesley.

[53] Szymanski, T. 1995. "Hypermeshes: Optical Interconnection Network for Parallel Computing," *Journal of Parallel and Distributed Computing*, Vol. 26, No. 1, pp. 1-22.

[54] Trivedi, K.S., 1982. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Prentice Hall.

[55] Warren, M. 1997. "Parallel Supercomputing with Commodity Components," *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, p. 1372.

[56] Willick, D.L., and D.L. Eager, 1990. "An Analytic Model of Multistage Interconnection Networks," *ACM SIGMETRICS*, pp 192-202.

[57] Wilson, J. 1997. "Interesting Problems; Potential Applications," *IEEE Computer*, Vol. 30, N. 10, pp 34-35.

[58] Ziavras, S. 1996. "A Low-Complexity Parallel System for Gracious, Scalable Performance Case for near Petaflops Computing", *Proceedings of the IEEE 1996 Symposium on the Frontiers of Massively Parallel Computing*, pp. 363-370.